UBISECURE™

Connecting Identity.
Transforming Digital Business.

SAML vs OAUTH 2.0 vs OPENID CONNECT

Understanding the differences between the three most common authorisation protocols

UBISECURE™

Contents

Introduction 3

- History 3

Acronyms & Terminology 4

- Web Single Sign-On 4

- Applications and Protected APIs 4

- Acronyms 5

Authorisation Protocols 5

- OpenID 2.0 5

- OAuth 2.0 6

- SAML 2.0 8

- OAuth 2.0 Extensibility 9

- OpenID Connect 1.0 9

Security Considerations 11

- SAML 11

- OAuth 2.0 and OpenID Connect 1.0 12

Comparing the Protocols 13

Conclusions 14

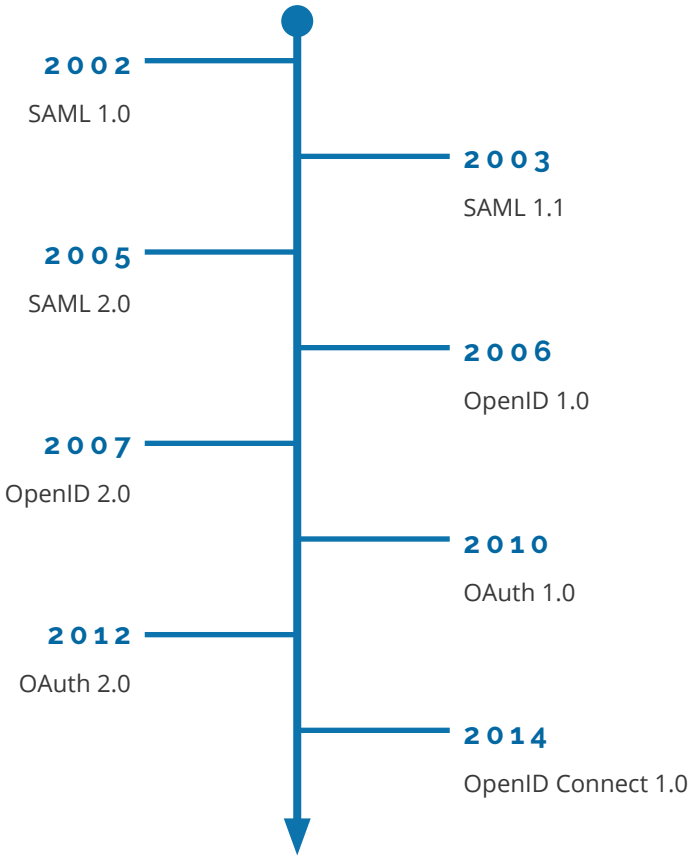
Introduction

The world of Identity and Access Management is ruled by two things - acronyms and standards.

In our popular blog post at https://www.ubisecure.com/uncategorized/difference-between-saml-and-oauth/ we compared the two most common authorisation protocols - **SAML2** and **OAuth 2.0**. This white paper extends that comparison with the inclusion of a third protocol, **OpenID Connect**. We also touch on the now obsolete OpenID 2.0 protocol.

HISTORY

Before diving into the details, it is useful to understand the order of emergence of the various protocols. It helps follow the evolution that gave rise to OpenID Connect.



UBISECURE™

Acronyms & Terminology

Let's recap on the terminology that will be needed as we start looking at the operation of the various protocols.

WEB SINGLE SIGN-ON

Party	Term in SAML	Term in OAuth	Term in OpenID
Web browser that an end user uses to access a web application	User agent	User agent	User agent
Server that owns the user identities and credentials	Identity Provider (IDP, IdP)	Authorisation Server (AS)	OpenID Provider (OP)
Web application that requires permission to proceed or access a resource	Service Provider (SP)	Client	Relying Party (RP) or Client
Server that hosts the resource being accessed	Service Provider (SP)	Resource Server	Resource Server

APPLICATIONS AND PROTECTED APIS

Party	Term in OAuth
Server that owns the user identities and credentials	Authorisation Server (AS)
Application that wants to access a protected API	Client
Protected API	Resource Server (RS)

ACRONYMS

SAML	Security Assertion Markup Language
OAuth	Open Authentication
OP	OpenID Provider
SP	Service Provider
RP	Relying Party
AS	Authorisation Server
HTTP	Hypertext Transfer Protocol
OIDC	OpenID Connect
SSO	Single Sign On
CIAM	Customer Identity and Access Management
XML	Extensible Markup Language
API	Application Program Interface

Authorisation Protocols

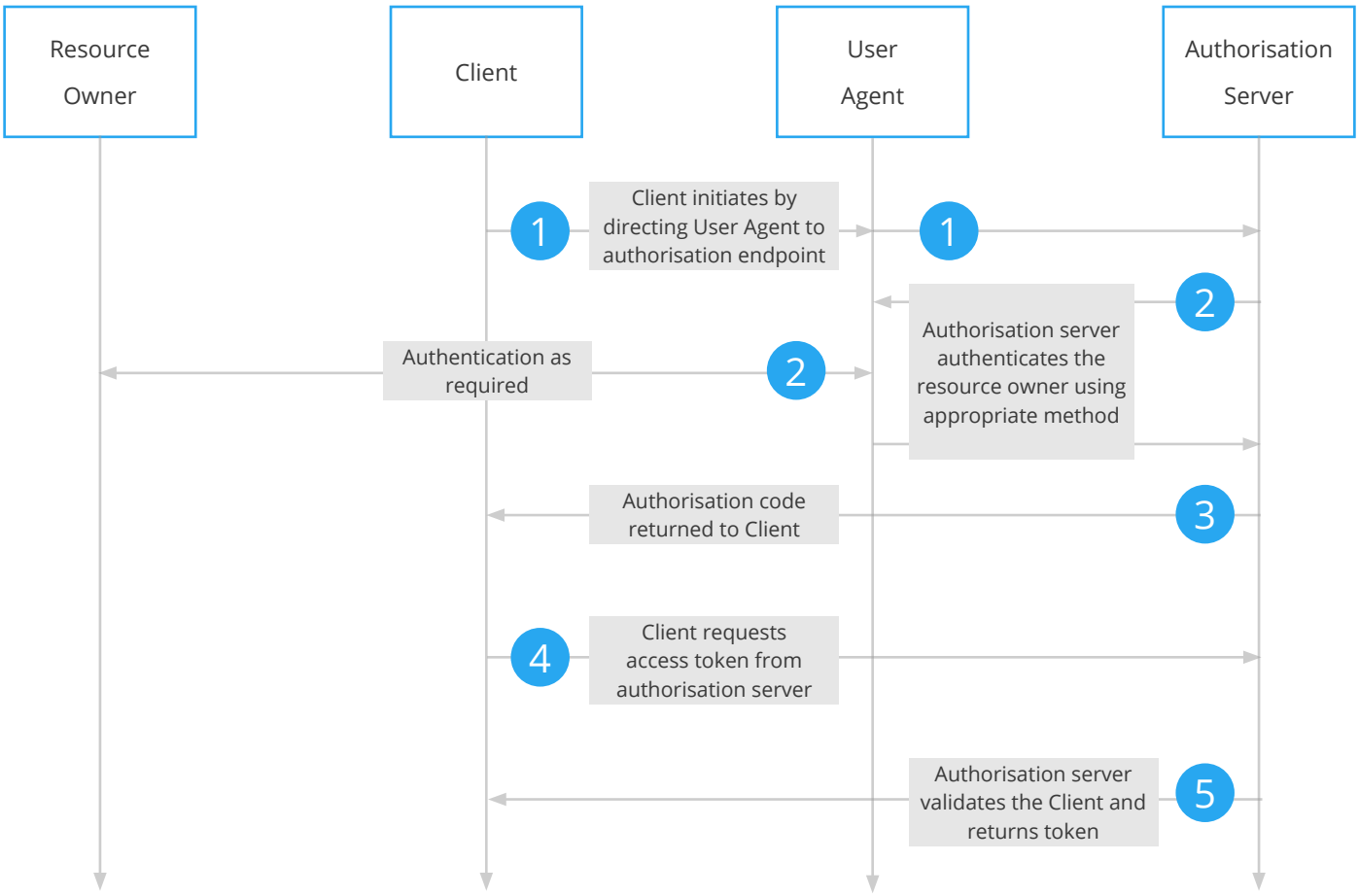
OPENID 2.0

OpenID was the first mainstream standard for authentication. It is, however, now obsolete following the approval of OpenID Connect. OpenID 2.0 was widely used and supported by most large internet companies.

OpenID provided user authentication and, with extensions in 2007, user attributes.

Today when 'OpenID' is being talked about, it will almost always refer to OpenID Connect 1.0.

OAUTH 2.0



The flow illustrated includes the following steps:

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

2. The authorisation server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.

3. Assuming the resource owner grants access, the authorisation server redirects the user-agent back to the client using the redirection URI provided earlier (in the request or during client registration). The redirection URI includes an authorisation code and any local state provided by the client earlier.

4. The client requests an access token from the authorisation server's token endpoint by including the authorisation code received in the previous step. When making the request, the client authenticates with the authorisation server. The client includes the redirection URI used to obtain the authorisation code for verification.

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

Authentication is all about the user in the context of the application, and a network authentication protocol like OpenID is able to do this across networks and security boundaries. An authentication protocols tells an application who the current user is and whether or not the user is present. In addition, the protocol will often return additional attributes, e.g. an e-mail address.

However, OAuth 2.0 tells the application none of those things. OAuth 2.0 tells it absolutely nothing about the user's identity, nor does it expose how the end user proved their presence or even if the user is still present or not. An OAuth 2.0 client asked for a token, received a token, and used that token to access some resource – for example, an API.

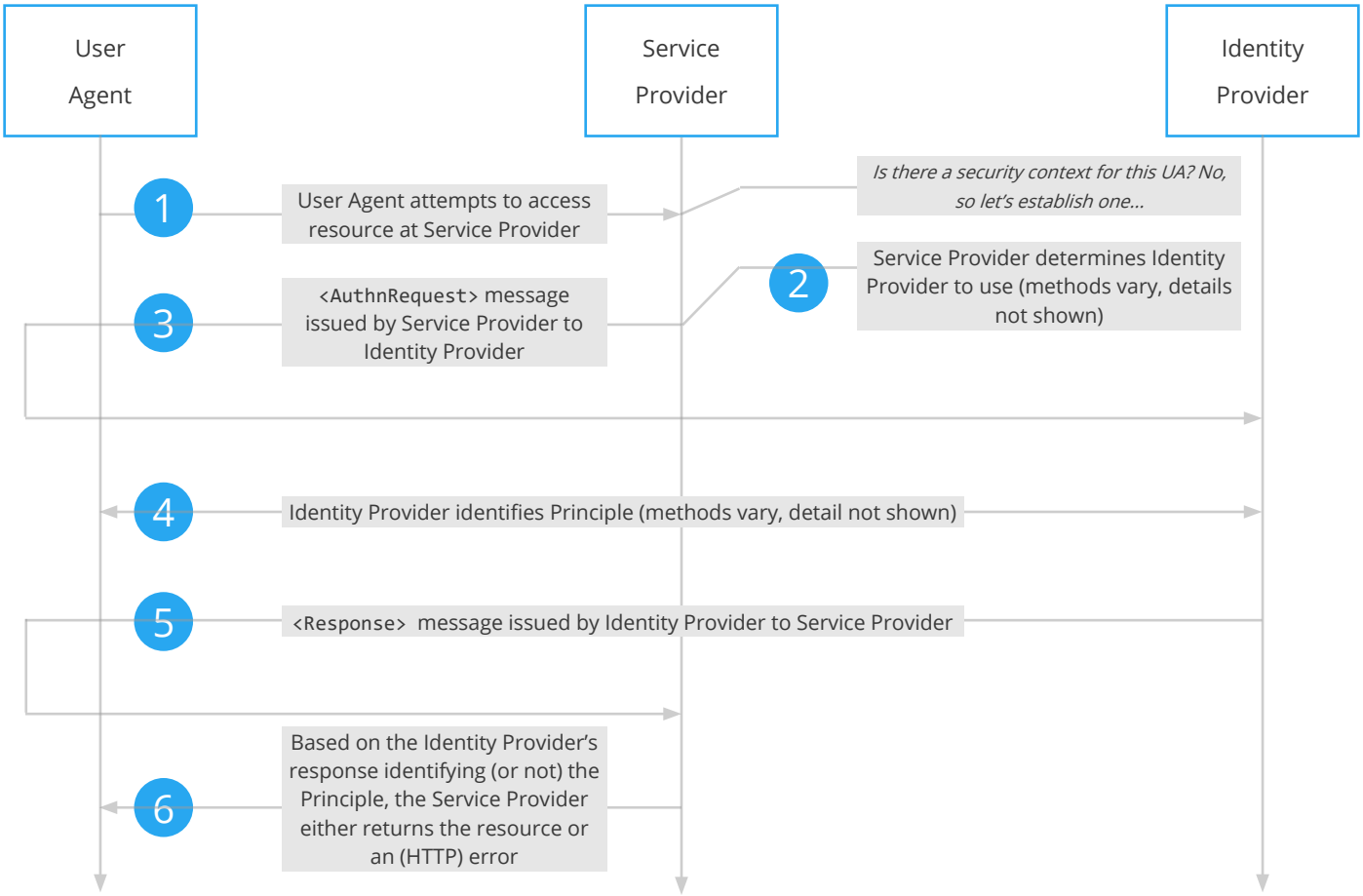
This delegated access, accessing a resource on the behalf of a user who may not even be present, is one of the great points of OAuth 2.0 and great for client authorisation. At the same time, it is very limited for authentication, where the whole point is figuring who the user is, how the user's identity was authenticated and whether the user is present or not.

The authorisation code grant type is used to obtain both access tokens and refresh tokens and is optimised for confidential clients. Since this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorisation server.

Note: the lines illustrating steps 1, 2, and 3 are broken into two parts as they pass through the user-agent.

SAML 2.0

We have seen that OpenID 2.0 provides authentication information, OAuth2 provides authorisation information. SAML2 provides all of this in the form of assertions (be they authentication assertions, attribute assertions or authorisation assertions).



1. The Client attempts to access a secured resource via the User Agent, without a security context.

2. The Service Provider obtains the location of an endpoint at an identity provider for the authentication request protocol. The means by which this is accomplished is implementation dependent.

3. The Service Provider issues an <AuthnRequest> message to be delivered by the user agent to the Identity Provider.

4. The Identity Provider authenticates the Client using some (undefined) method.

5. The Identity Provider issues a <Response> message to be delivered by the User Agent to the Service Provider. The message may indicate an error, or will include (at least) an authentication assertion.

6. Having received the response from the Identity Provider, the Service Provider can respond to the Client's User Agent with its own error, or can establish its own security context for the Clienty and return the requested resource.

This flow is HTTP POST heavy, rending it only really useful for interactive web sessions.

OAUTH 2.0 EXTENSIBILITY

Before jumping to OpenID Connect, it is useful to understand a little about OAuth2 extensibility. As part of the specification work on OAuth2, the definition was created with extensibility in mind. This extensibility allows new protocols to be defined on top of the 'basic' OAuth2 protocol.

A number of extension protocols have been defined (for example UMA) and a number more are in draft stages.

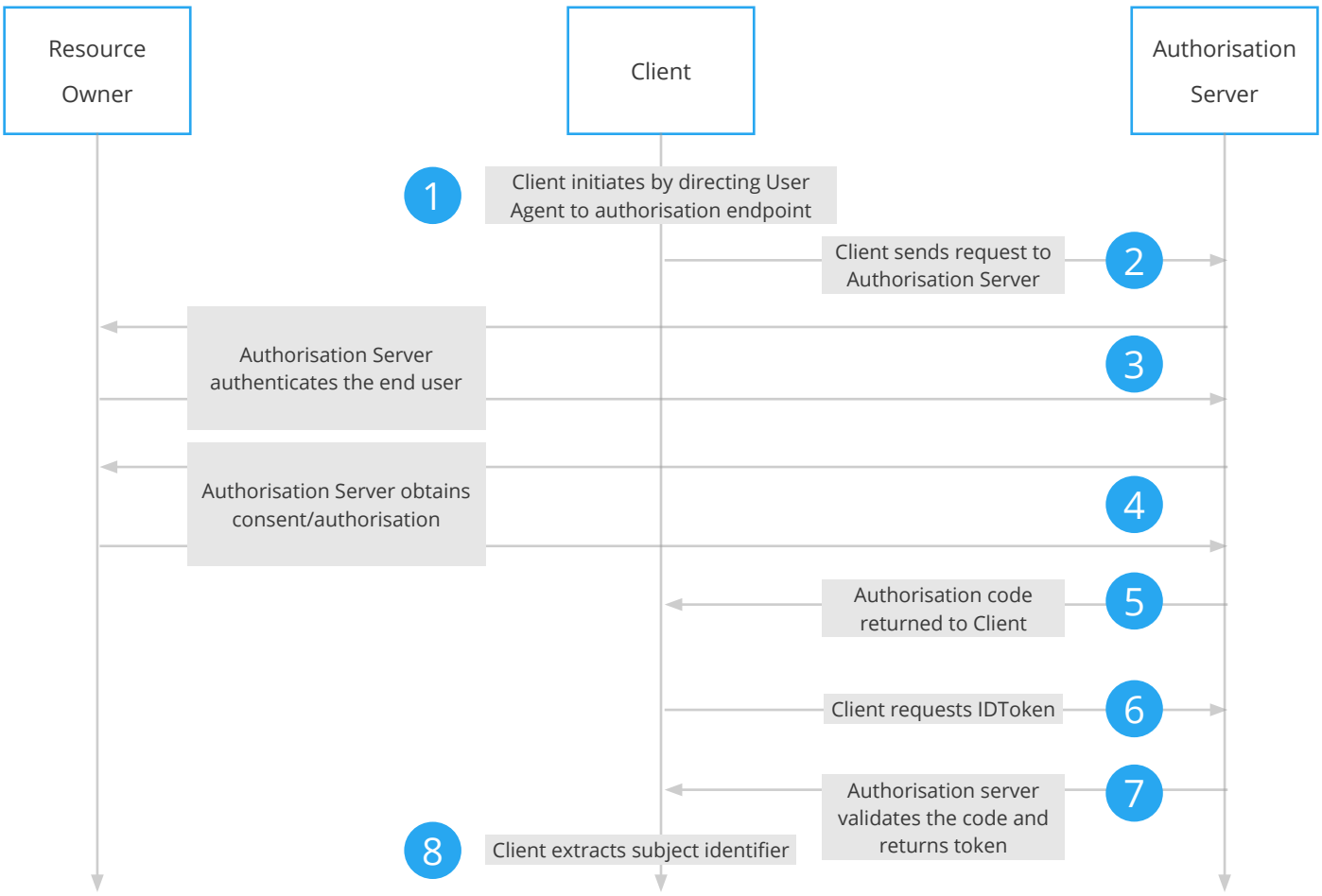
Of specific note here is OpenID Connect, a protocol built on top of OAuth2 to provide: "authentication built on top of OAuth 2.0 and the use of claims to communicate information about the End-User". [1]

1 http://openid.net/developers/specs/

OPENID CONNECT 1.0

OpenID Connect, or OIDC for short, layers on top of OAuth 2.0 to provide authentication information, as well as authorisation information, through the use of Claims.

Being based on OAuth2.0, OpenID Connect flow is very similar to OAuth 2.0 (as seen below). The major difference is the addition of the ID Token.



1. The client prepares an Authentication Request containing the desired request parameters.

2. The client sends the request to the Authorisation Server.

3. The Authorisation Server authenticates the end user.

4. The Authorisation Server obtains user consent/authorisation.

5. The Authorisation Server sends the user back to the client with an access token and, if requested, an ID token.

6. The client requests a response using the Authorisation Code at the Token endpoint.

7. The client receives a response that contains an ID Token and Access Token in the response body.

8. The client validates the ID token and retrieves the subject identifier of the user.

The OpenID Connect standard introduces additional parameters to the request and introduces the concepts of Claims and the ID Token.

Feature	Details
Additional parameters	When the Client makes a request to the IdP it can pass additional parameters to the IdP, for example language for the authentication UI, and required authentication level.
Claims	Claims are 'name-value' pairs that provide attribute data about the user that has been authenticated. The OpenID Connect standard defines a number of standard Claims, for example: given_name, family_name, picture, email. In addition to the standard Claims, 'additional claims' can be defined.
ID Token	The ID Token is built by the IdP and returned to the Relying Party as the final part of the basic authentication. This token contains a number of authentication related claims and may contain additional user related claims. The token itself is in the form of a signed JSON Web Token (JWT), and may, optionally, be encrypted. An access token is also returned that can be used by the Client to make further requests to the IdP or a Resource Server.

Again, being based on OAuth 2.0 OpenID Connect is 'API friendly' and can be used by web applications, desktop applications, mobile applications and devices.

Security Considerations

SAML

For SAML the most common and widely used protocol for securing message exchange between an IdP and service provider is sending signed SAML Assertion to the Service Provider using HTTP POST protocol. SAML uses XML Signature and XML Encryption for end-to-end message integrity and encryption.

Key exchange in SAML happens either out of band or dynamically with SAML Metadata.

See SAML V2.0 Implementation Profile for Federation Interoperability for a recent requirement specification for an interoperable and secure SAML implementation.

OAUTH 2.0 AND OPENID CONNECT 1.0

OIDC defines different profiles for securing message exchange between an IdP and a Relying Party.

The simplest model completely relies on security of DNS and TLS for integrity. The benefit is a very low barrier of entry into integrating an application with an OIDC IdP where the application designer only needs capability of invoking a RESTful API of the OIDC IdP - no processing of digital signatures or other cryptography is necessary.

By implementing signed and encrypted JWTs, the security of an OIDC implementation is increased to the level of SAML with end-to-end message integrity and encryption.

By using modern technologies such as Token Binding, security increases even beyond what is possible with standards based interoperable SAML.

When signed and encrypted JWTs are used, key exchange in OIDC happens either out of band or dynamically with OIDC Metadata.

Dynamic Client Registration Protocol enables a very dynamic and completely automated management of Client identities at an OIDC IdP - very well suited for IoT and Mobile Application use cases.

See OAuth 2.0 Threat Model and Security Considerations and OAuth 2.0 Security Best Current Practice for recent instructions and best practices into creating secure OAuth and OIDC integrations.

Comparing the Protocols

Firstly, OAuth 2.0 has a different purpose to both SAML 2.0 and OpenID Connect 1.0, OAuth is, at the base level, an authorisation protocol, whereas SAML and OpenID Connect are authentication/authorisation protocols.

For access control, OAuth 2.0 provides a great solution.

SAML and OpenID Connect both provide authentication as well as authorisation. SAML is definitely the more complex to implement. OpenID Connect, being based on OAuth has a very low barrier to entry and can be scaled once working (both security and feature wise).

If you are looking to join large existing federations then SAML will have the edge, its age means many of the existing federations are SAML based (for example, most university federations). Of course, it is possible to translate between protocols. Doing this manually is a significant amount of work, but if you're using the Ubisecure Identity Server in an IdP Proxy role this is just a configuration option, so it is possible to have a simple implementation of OpenID Connect working with an existing SAML federation.

SAML is effectively constrained to browser operation, so for application or device usage OpenID Connect will be the protocol of choice.

When it comes to security, OAuth, and hence OpenID Connect, provides a flexible model which can scale. SAML benefits from being within the browser and having a relatively fixed security model, but is also only as secure as the browser. OAuth is dependent upon the TLS stack and, when incorporated in an application, is only as good as the stack in the application. If the app is using the OS implementation then it will be parallel to the browser. However, device implementation will need to take care over stack selection and configuration.

Conclusions

The emergence and rapid growth of OpenID Connect is a physical manifestation of the ongoing mobile transformation. As identity and access management has split between the traditional enterprise SSO and external user centric CIAM, so has the protocol stack underneath. OpenID Connect combines OAuth 2.0 authorisation with authentication, and allows building a user-friendly mobile application “done right”.

An inevitable question is “Which of the protocols is the right for my use case?” Making broad generalisations is one of the best ways to draw dissension from all directions, but the below table does just that.

Protocol	Best use cases
SAML	Enterprise SSO, existing federations
OAuth 2.0	API authorisation, UMA
OpenID Connect	Customer SSO, CIAM, mobile

If you are reading this to determine how to quantify, manage and reap the benefits of CIAM in your organisation please contact Ubisecure – we can guide you through the complexity and reduce your implementation time considerably.

www.ubisecure.com
sales-team@ubisecure.com

About Ubisecure

Ubisecure is a pioneering European b2b and b2c Customer Identity & Access Management (CIAM) software provider and cloud identity services enabler dedicated to helping its customers realise the true potential of digital business. Ubisecure provides a powerful Identity Platform to connect customer digital identities with customer-facing SaaS and enterprise applications in the cloud and on-premise. The platform consists of productised CIAM middleware and API tooling to help connect and enrich strong identity profiles; manage identity usage, authorisation and progressive authentication policies; secure and consolidate identity, privacy and consent data; and streamline identity based workflows and decision delegations. Uniquely, Ubisecure's Identity Platform connects digital services and Identity Providers, such as social networks, mobile networks, banks and governments, to allow Service Providers to use rich, verified identities to create frictionless login, registration and customer engagement while improving privacy and consent around personal data sharing to meet requirements such as GDPR and PSD2.

Ubisecure is accredited by the Global Legal Entity Identifier Foundation (GLEIF) to issue Legal Entity Identifiers (LEI) under its RapidLEI brand, a cloud-based service that automates the LEI lifecycle to deliver LEIs quickly and easily. The company has offices in London and Finland.

UBISECURE®

www.ubisecure.com
sales@ubisecure.com