OpenID Connect

A HIGHLY SECURE AUTHENTICATION PROTOCOL FOR RESTFUL APIS

Dipl. Math. Xenia Bogomolec
and
Dipl. Inform. Peter Rosemann
24.04.2017

Content

- Basic Concept
  - ID Providers and Relying Parties
  - OpenID Connect/OAuth 2.0
  - Three Tokens
- Client Types
- Security Considerations
  - OpenID Connect instead of conventional Authentication
  - Levels of Security
- Mobile Applications
  - Comparison of iOS and Android
- Integration in Clouds

Basic Concept

ID Providers and Relying Parties

An ID provider is an instance who holds user accounts. It saves the credentials of a user and can therefore authenticate him.

A relying party is an application that asks the ID provider to ensure them that the user authenticated himself. It does not hold the credentials of the user. Classically it is a third party application, which means, that it is owned by someone else than the ID/account provider.

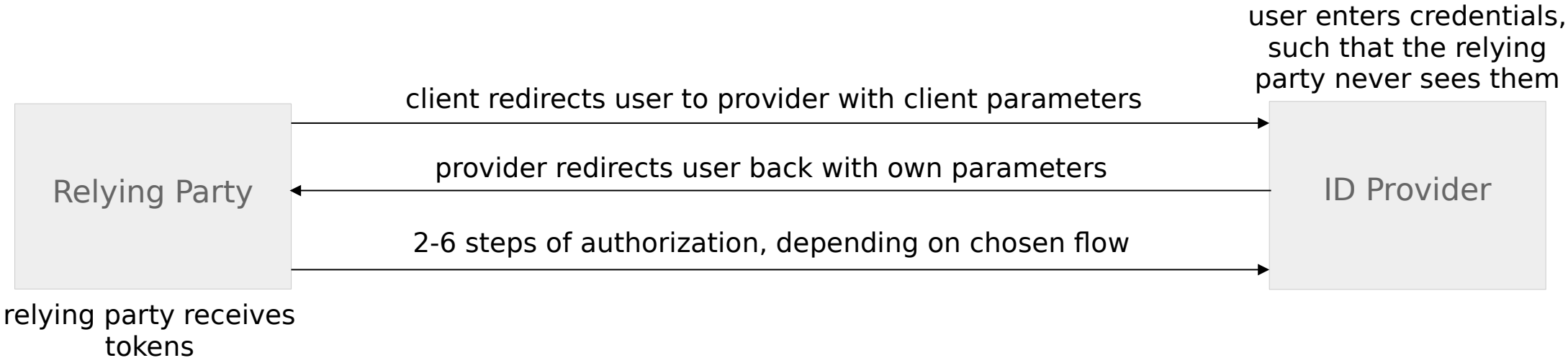
OAuth 2.0 and OpenID Connect are originally developed to solve the authentication of users for third party applications. Therefore the authorization of the third party application is bound to the authentication process.

For a REST API, which is designed to be connected to various kinds of clients (e.g. browser based, native) it makes sense to think about binding all authentication processes to an OAuth-based protocol. Like this the risk of phishing and man in the middle can be mitigated considerably.

Basic Concept

OpenID Connect and OAuth 2.0

OpenID Connect is an authentication protocol that sits on top of OAuth 2.0. The pattern of OAuth 2.0 is described here:



Basic Concept

OpenID Connect and OAuth 2.0 / Common Ground

A common characteristic of the login process for OpenID Connect and OAuth 2.0 is the redirection of the user to the ID providers login site.

If the authentication of the user is successful, it is followed by an authorization process between the ID provider and the relying party.

NOTE:
For a confidential client an authentication of the relying party with a client secret is included too.

Basic Concept

OpenID Connect and OAuth 2.0 / Differences

OAuth 2.0 is a protocol that provides authorization to access a protected resource. The ID provider can authenticate this user for other applications and authorize them to access the protected resource of this user. It issues two kinds of tokens, an access token and a refresh token.

OpenID Connect is a protocol that provides authorization to retrieve information about a user. It is sitting on top of OAuth 2.0. The ID provider can authenticate this user for other applications and get extra information about this user. It adds an ID token to the tokens of OAuth 2.0.

Availability of features:

	resource access	user info	access token	refresh token	ID token
OAuth 2.0	✓	✘	✓	✓	✘
OpenID Connect	✓	✓	✓	✓	✓

OpenID Connect also provides more preconfigured types of authorization flows than pure OAuth 2.0.

Basic Concept

Three Tokens

The three tokens are issued after a successful authentication and authorization. Each one is user- and client-application-bound and serves a special purpose:

Access token:
Grants the user access to a protected resource. It is sent in the HTTP authorization header with each request. Its time of validity can be compared to a conventional session length. A default value might be an hour.

Refresh token:
Is sent by the relying party to the ID provider to get a new access token for a user. It is usually valid for a long term, e.g. half a year. This mechanism allows a user not having to re-login unless he does not use the application for a longer time. The re-login is handled by the relying party and the ID provider in the background.

ID token:
A JSON web token that contents user info and information about the issuer of the token. It is only valid for a very short time, e.g. 10 minutes. The ID provider can be verified by its signature.

Client Types

There are basic differences between relying parties that

- can keep a client secret, e.g. another web server
- cannot keep a client secret, e.g. a simple Javascript client
- can keep other kind of data safe to a certain extent, e.g. a mobile application

A relying party that can store a client secret in a safe way is called a confidential client. A relying party that cannot store a client secret is called a public client. There are several authorization flows available for the various kinds of client types.

Security Considerations

OpenID Connect instead of Conventional Authentication for REST APIs

Basically open API endpoints can be connected to by any source from a network. Because of this, such endpoints are always a target for attackers.

The protocols including authorization provide the extra security feature of verifying where a login or registration request is coming from. This mitigates the risk of phishing attacks.

It is a consequential REST'ful manner to include the users ID in the path to the users resource. This is a basic user info that uniquely identifies the user to the relying party. It is delivered by the ID token or at the user_info-endpoint of the ID provider.

OpenID Connect satisfies the challenges of authentication and authorization in the scope of a REST API. Even if there are no third party applications involved, it offers a highly secure and elegant solution for the authentication of users from all connecting clients (mobile applications etc.).

Security Considerations

Levels of Security

The various flows and client types offer different kinds of security levels. They are based on the security that a relying party or an architecture of a system can provide.

The most secure relying party

… is an application that is embedded in a secure environment itself, like a web server. Such parties can store an client secret, which is sent to the ID provider in the HTTP-Authorization header during the OAuth 2.0 process.

The least secure relying party

… is an application that cannot store a client secret itself, like a simple Javascript client. For the according flow, which is called implicit flow, it is recommended to add additional security measurements. For a REST API which only serves own applications this flow can easily be avoided.

Something in between

… is an application that is embedded in a partially secure environment, like an android mobile phone. Such parties cannot store a client secret in secure way, but they can keep a refresh-token.

Mobile Applications
Comparison of iOS and Android

OpenID connect offers a secure solution for a centralized authentication for mobile applications. The security also depends on the handling of the communication and the parameters outside of the chosen protocol. A login should happen via the WebView (or external WebBrowser) to hide any credentials from the native app. After the successful login at the OpenID provider the website will send a code to the native app via a specific callback URI. The app can request the access and refresh token with the code plus some more parameters. Within this configuration, the top level security flow can be applied. Access token will expire after a specified time. The app requests a new access token with the provided refresh token.

An iOS app for example
… can use the highets level of security. Access & refresh token are stored in the keychain. The Callback URI must be configured in Xcode in the URL Type section. Client ID, client secrect and some more parameters are hardcoded in a compilable file (NOT .plist or other resource file). Any compiled files are encrypted by Apple when submitting to the App Store.

An Android app in contrary
… has to be considered less secure compared to an iOS environment. Android does not offer a built-in keychain. Secure data (access & refresh token) must be encrypted by 3rd party libraries. The so called hybrid flow could be a solution for the authentication and authorization.

Integration in Clouds

If OpenID Connect is used for applications served from platforms likes AWS, which also has its own authentication system, it has to be carefully considered on how to combine the mechanisms.

If for example an OpenID Connect token of a user is exchanged for an AWS token, the securing mechanisms of the OpenID Connect Protocol might be broken. The direct and secured communication between the ID Provider and the instance who handles the access to protected resources are the only way to ensure that

- The ID provider knows that the relying party is who it claims to be
- The relying party knows that the tokens are really issued by the ID provider
- A man in the middle attack is highly unlikely
- The comfort of the silent re-login via the refresh token is secure

Contributors

Various Views on a Complex Subject

Initiator and OpenID Connect Evangelist

Peter Rosemann

Author of Slides

Xenia Bogomolec (algorithmic protocols, technical networks and math)
(Assigned to the integration of OpenID Connect in the context of online payment services by DieDefa GmbH)

Thanks for the Input of the Contributors

Christian Menschel (iOS engineer)
Alexander Block (devop, system architecture and technical networks)
Thomas Krüger (highly experienced full stack web developer