Gluu Blog

Follow us:

Back to Blog

OAuth vs. SAML vs. OpenID Connect

Mike S.
December 7, 2016

[:en]

Today there are three dominant open web standards for

Subscribe

identity online: OAuth, SAML and OpenID Connect. In the following article we'll examine how the technologies relate to each other, and under which circumstances each technology should be used.

OAuth 2.0 vs. OpenID Connect

The first thing to understand is that OAuth 2.0 is an authorization *framework*, not an authentication protocol. OAuth 2.0 can be used for a lot of cool tasks, one of which is person authentication.

OpenID Connect is a "profile" of OAuth 2.0 specifically designed for attribute release and authentication.

From a technical perspective, the big difference between OpenID Connect and OAuth 2.0 is the `id_token`–there is no `id_token` defined in OAuth 2.0 because it is specific to federated authentication.

The `id_token` provides an additional layer of security to user sign in transactions by adding:

A nonce, which is sent by the client and enables the integrity of the response to be validated;

A hash of the access token;

A hash of the code (optional).

Net-net, OpenID Connect is laser-focused on user authentication, whereas OAuth 2.0 was left generic so it could be applied to many authorization requirements, like API access management, posting on someone's wall, and using IOT services.

SAML vs. OpenID Connect

At the risk of over-simplification, OpenID Connect is a rewrite of SAML using OAuth 2.0. Let's look at a few similarities and differences…

IDP / SP vs. OP / RP

In SAML, the user is redirected from the Service Provider (SP) to the Identity Provider (IDP) for sign in.

In OpenID Connect, the user is redirected from the Relying Party (RP) to the OpenID Provider (OP) for sign in.

The SAML SP is *always* a website. The OpenID Connect RP is either a web *or* mobile application, and is frequently called the "client" because it extends an OAuth 2.0 client.

In both cases, the IDP/OP controls the login to avoid exposing secrets (like passwords) to the website or app.

Assertion vs. id_token

In SAML, there is an "assertion"–a signed XML document with the subject information (who authenticated), attributes (info about the person), the issuer (who issued the assertion), and other information about the authentication event.

The equivalent in OpenID Connect is the id_token. This is a signed JSON document that contains the subject, issuer, and authentication information.

Front Channel vs. Back Channel

A big difference between OpenID Connect and SAML is the use of “front-channel” and “back-channel”:

> The *front-channel* is the browser;
>
> The *back-channel* is communication directly between the application and the IDP/OP.

Although SAML defines back-channel mechanisms, they are rarely used in practice. The most common way SAML sends the request XML and response XML (assertion) is via the browser. Most SAML sites use the “POST Binding” to send the response.

In this scenario, the browser is sent an HTML form from the IDP with the response XML as a form parameter.

There is some JavaScript in the form to auto-submit the data back to the SP. It's a neat trick because the SP and IDP don't need network connectivity to communicate–the browser acts as a middle-man!

OpeniD Connect defines a similar mechanism ("Form Post Response Mode")–but unlike SAML, it's use is more the exception than the rule. Both OpenID Connect and SAML frequently use something like the "Redirect Binding" to send the request. This is where the URL parameters are used to send the XML. This also leverages the browser.

OpenID Connect normally uses the back channel–a direct call from the RP to the OP–to retrieve this information. The attributes (or "user claims" in OpenID jargon) are available to the client by calling the user_info endpoint, which is a JSON REST API. However, because this is OAuth 2.0, the client needs a token to call this API. And according to the OAuth 2.0 framework, the

token is obtained from the OPs token endpoint using the back channel.

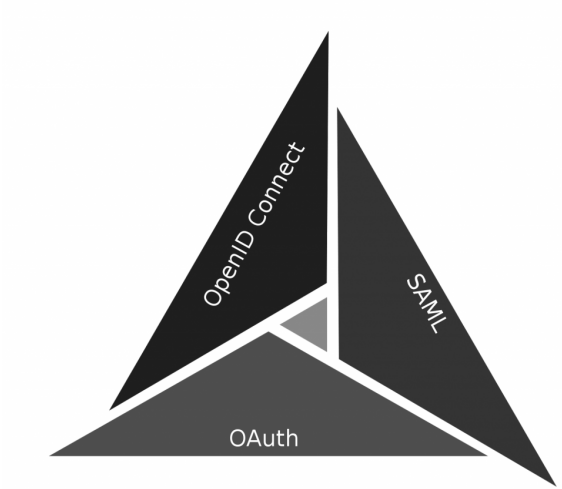
Which protocol, when?

So when should SAML be used, and when should OAuth 2.0 or OpenID Connect be used instead?

> Mobile applications: no question–use OpenID Connect.
>
> If the application already supports SAML: use SAML.
>
> If you are writing a new application, use OpenID Connect–skate to where the puck is going!
>
> If you need to protect API's, or you need to create an API Gateway… that's a topic for another blog. Short answer: use OAuth 2.0 or the User Managed Access ("UMA") protocol.

The good news is that if you are using the Gluu Server, you'll be able to support single sign-on (SSO) whether your apps use SAML, OpenID Connect or OAuth 2.0.

Have questions? Just schedule a call.[:ja]



Gluu Serverは、SAMLとOAuth2両方のコンポーネントを持つ無料のオープンソース・プラットフォームです。 私は二つのインフラストラクチャーの相違について教育活動をコミュニティで行っており、 ここでは、理解を深めるための簡単な概要を紹介します。

OAuth 2.0は、認可フレームワークであり、認証プロトコルではありません。 このフレームワークは認可の公分母と考えることができます。 OAuthは、Webまた

はモバイルサインインの流れの定義以上のさまざまなユースケースを解決するのに使うことができます。OAuthの文脈でのSAML相当のものを探している場合は、OpenID Connect（以下、Connectと表記）を詳しく見ていく必要があります。

過度の単純化のリスクはありますが、ConnectはOAuth2版SAMLといえます！ 類似点のいくつかを見てみましょう。

SAMLでは、ユーザーはWebサイトから「IDP」（IDプロバイダ）にリダイレクトされます。 Connectでは、ユーザーはWebサイト（またはモバイルアプリ）から「OP」（OpenIDプロバイダ）にリダイレクトされます。 どちらの場合も、ウェブサイト/アプリに（パスワードのような）秘密が漏れるのを回避するために、IDP | OPがログインを制御します。

SAMLには“SP”（サービスプロバイダ – 常にウェブサイト）があります。 コネクトでは、「RP」（信頼関係者 – ウェブサイトまたはモバイルアプリケーション）があります。 RPは、OAuth 2.0クライアントを拡

張しているため、「クライアント」と呼ばれることもあります。

SAMLには、サブジェクト情報（被認証者）、属性（当該個人に関する情報）、発行者（アサーション発行者）、および認証イベントに関するその他の情報を含む署名付きXML文書である「アサーション」があります。 OpenID Connectにおいて同様の役割を果たすのはid_tokenです。 これは、サブジェクト、発行者、および認証情報を含む、署名されたJSONドキュメントです。 ここに属性は入っていませんが、これについては後ほど説明します。

ConnectとSAMLの大きな違いは、「フロントチャネル」と「バックチャネル」の使用です。フロントチャネルはブラウザです。 バックチャネルは、WebサイトとIDP間の直接の通信です。

SAMLではバックチャネルメカニズムが定義されていますが、実際にはほとんど使用されていません。 SAMLが要求XMLと応答XML（アサーション）を送信する最も一般的な方法は、ブラウザ経由です。 ほとん

どのSAMLサイトでは、応答には「POST Binding」を使用しています。 このシナリオでは、ブラウザには、応答XMLをフォームパラメーターとしてIDPからHTMLフォームが送信されます。 このフォームには、SPに自動送信するためのJavaScriptが含まれています。 これは、SPとIDPの間にネットワーク接続を必要としないためのうまい策略です。ブラウザは中間者として振る舞います。

Connectも同様のメカニズム（”Form Post Response Mode”）を定義していますが、SAMLとは異なり、その使用はルールというより例外です。 Connect、SAML双方ともに、リクエストを送信するために「リダイレクトバインド」のようなものを（頻繁に）使用します。 この場合、XMLの送信にはURLパラメータが使用されます。 これもまた、ブラウザを利用します。

さて、それでは先程先送りにした属性について話しましょう。Connectは通常、属性情報を入手するためにバックチャネル-RPからOPに直接呼び出しを使用します。 クライアントは、user_info endpointというJSON

REST APIを呼び出すことによって、属性（またはOpenIDの専門用語で「ユーザークレーム」）の提供をうけることができるのです。 しかし...このAPIはOAuth2準拠なので、クライアントは呼び出すためにトークンを必要とします。 また、OAuth2フレームワークによれば、OPのトークンエンドポイントからバックチャネルを使用してトークンは取得されます。

OpenIDとOAuth2の間の大きな違いはid_token　です。
id_tokenはOAuth2では定義されていません。というのは、id_tokenは SAMLアサーションのようなものでフェデレーション認証に固有のものだからです。

id_tokenは、いくつかの付加的なセキュリティ機能を備えています。

> クライアントによって送信されたNonce（ランダム値）を含み、応答の完全性を検証することができます。

アクセストークンのハッシュが含まれています。

オプションで、コードのハッシュを含みます。

これらの機能は、プレーンなOAuth2上での追加のセキュリティ層を提供します。 さらに、OpenID Connectは、SAML SPから署名され暗号化された要求と同様に、署名され暗号化された要求オブジェクトを送信する方法を定義します。

OAuth2は、APIアクセス管理や、掲示板への投稿、IOTサービスの使用など、多くの認可要件に適用できるように汎用的なままにしました。それは良いことです！ いろいろなクールなタスクのにOAuth2は使用できるのです。個人認証は、そのうちの1つです。

So when should you use SAML, and when should you use OpenID Connect?

いつSAMLを使用すべきですか、いつOpenID Connectを使うべきですか？ モバイルアプリケーションをお持ちの場合は、疑い無くOpenID Connectを使用してください。AppAuth利用に関するブログ記事を　参照してください

既にSAMLを使用しているアプリケーションを使用している場合はSAMLを使用してください！

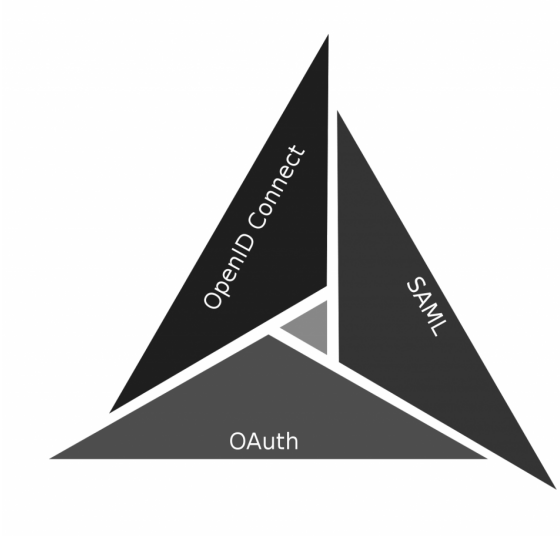
新しいアプリケーションを書いているなら、OpenID Connectを使いましょう。（アイスホッケーでは）パックが行く方へスケートするものです！（訳注：群れ（パック）が行く方へ向かえという慣用句とのダジャレ。）

> APIを保護する必要がある場合や、APIゲートウェイを作成する必要がある場合は...これは、別ブログのトピックですね。一言で言うなら：OAuth2またはUser Managed Access (“UMA”)　プロトコルを参照してください！

この記事が役に立つことを祈ります。良い知らせがあります。もし、あなたがGluuサーバを使っているならば、アプリケーションがSAMLを使っているかOpenID Connectを使っているかにかかわらず、シングルサインオン(SSO)させることができます。楽しくSSOしましょう！

このことについて質問がありますか？Gluuの製品とサービスに興味がありますか？その場合は、このリンクからわたしとの電話会議を設定してください。

[:es]



Gluu Server es una plataforma open source gratuita que tiene componentes SAML and OAuth2. Estuve tratando de ayudar a educar a la comunidad por un buen tiempo en los pros y contras de ambas infraestructuras. Esta es una breve introducción para ayudarlo a orientarse!

OAuth 2.0 es un framework de autorización, no un protocolo de autenticación. Se puede pensar en este framework como un denominador común para autorización. OAuth se puede utilizar para resolver diferentes casos de uso, no solo definir un flujo de inicio de sesión web o mobile. Si lo que está buscando es el equivalente OAuth para SAML, debe examinar más de

cerca OpenID Connect (en adelante, Connect para abreviar)

A riesgo de sobre-simplificar, Connect es OAuth2-SAML! Veamos algunas de las similitudes:

En SAML, el usuario es redirigido desde el website al "IDP" (Identity Provider). En Connect, el usuario es redirigido del website (o app móvil) al "OP" (OpenID Provider). En ambos casos, el IDP|OP controla el login para evitar exponer secretos (como passwords) al website/app.

En SAML tenemos un "SP" (Service Provider–siempre un website). En Connect tenemos un "RP" (Relying Party–un website o app móvil). Al RP se lo llama frecuentemente el "cliente" porque extiende un client de OAuth 2.0.

En SAML, existe un "assertion"–un documento XML firmado digitalmente con la información del sujeto (subject-que se autenticó), atributos (información

acerca de la persona), el emisor (issuer-que emitió el assertion), e información adicional acerca del evento de autenticación. El equivalente en OpenID Connect es el id_token. Este es un documento JSON firmado que contiene el sujeto, emisor e información de autenticación. Faltan los atributos, pero llegaremos ahí en un minuto.

Una gran diferencia entre Connect y SAML es el uso del "front-channel" y el "back-channel." El front-channel es el navegador. El back-channel es comunicación directa entre el website y el IDP.

Aunque SAML define mecanismos de back-channel, rara vez se usan en la práctica. La forma más común en que SAML envía la solicitud XML y la respuesta XML (aserción-assertion) es a través del navegador. La mayoría de los sitios SAML utilizan "POST Binding" para enviar la respuesta. En este escenario, se envía al navegador un formulario HTML del IDP con la respuesta XML como un parámetro de formulario. Hay algo de

JavaScript en el formulario para que se vuelva a enviar automáticamente al SP. Es un buen truco, porque el SP y el IDP no necesitan conectividad de red: ¡el navegador actúa como un intermediario!

Connect define un mecanismo similar ("Form Post Response Mode") – pero a diferencia de SAML, su uso es más la excepción que la regla. Tanto Connect como SAML (frecuentemente) utilizan algo como el "Redirect Binding" para enviar la solicitud. Aquí es donde se usan los parámetros de URL para enviar el XML. Esto también hace uso del navegador.

De vuelta a los atributos que faltan! Connect normalmente utiliza el back channel –una llamada directa desde el RP al OP– para recuperar esta información. Los atributos (o "user claims" en jerga OpenID) están disponibles al cliente llamando al endpoint user_info: una API REST JSON. Sin embargo… como esto es OAuth2, el cliente necesita un token para llamar a esta API, y de acuerdo al framework OAuth2, el

token también se obtiene utilizando el back channel– desde el token endpoint del OP.

La gran diferencia entre OpenID Connect y OAuth2 es el id_token. No hay un id_token definido en OAuth2 porque recuerde: el id_token es como un SAML assertion, es específico para la autenticación federada.

El id_token tiene algunos tiene algunas características de seguridad adicionales:

> Contiene un nonce, que fue enviado por el cliente y permite validar la integridad de la respuesta;
>
> Contiene un hash del access token;
>
> Opcionalmente, contiene un hash del código.

Estas características proporcionan una capa adicional de seguridad sobre el OAuth2 básico. OpenID Connect también define cómo enviar un request object cifrado y firmado, similar a una solicitud firmada y cifrada de un SP de SAML.

OAuth2 se dejó genérico para que pudiera aplicarse a muchos requisitos de autorización, como gestión de acceso a APIs, hacer un post en el muro de un usuario y utilizar servicios de IoT! ¡Eso es bueno! Puede utilizar OAuth2 para un montón de tareas interesantes, una de las cuales es autenticar personas.

Entonces, ¿cuándo debe usar SAML, y cuándo debe usar OpenID Connect?

> Si tiene una aplicación móvil, utilice OpenID Connect–sin dudarlo. Ver este blog acerca del uso de AppAuth (en inglés).

Si tiene una aplicación que ya soporta SAML, utilice SAML!

Si está escribiendo una nueva aplicación, use OpenID Connect, siga la tendencia!

Si necesita proteger API's, o necesita crear un API Gateway… es tema para otro blog. Respuesta corta: utilice OAuth2 o el protocolo User Managed Access ("UMA")!

¡Espero que esto sea de ayuda! La buena noticia es que si está utilizando el servidor Gluu, podrá dar soporte al inicio de sesión único (SSO) tanto si sus aplicaciones utilizan SAML como OpenID Connect. Diviértase haciendo SSO!

Tiene más preguntas? Está interesado en los productos y servicios de Gluu? Simplemente agende una llamada conmigo.

[:]

IAM oauth2 open source openid connect saml sso

Share:

Mike Schwartz

Mike has been an entrepreneur and identity specialist for more than two decades. He is the technical and business visionary behind Gluu. Mike is an application security expert and has been a featured speaker at RSA Conference, Gartner Catalyst, Cloud Identity Summity (now "Identiverse") and many other security conferences around the world.

11 Comments **Gluu Blog** **Login**

Recommend 5 **Share** **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Chris Skura • a year ago

How can you not like a computer guy who uses hockey terminology? awesome!!!

3 • Reply • Share ›

jonkernpa • 8 months ago

Thanks for clearing things up... Just as I was thinking there was a chance that OpenID Connect could be wielded like SSO, Google search served up your blog post.

2 • Reply • Share ›

Mike Schwartz • 2 years ago

Agreed, JWT's are great. That's why they are used extensively in the OpenID Connect spec. The question is what's in that JWT? What are the claims? How do you know that the JWT that you got back in the response relates to the request you sent (i.e. Connect defines the nonce)? How do you know that the access token has not been modified (i.e. Connect defines the at_hash)? How do you know the code has not been modified (i.e. Connect defines the c_hash). How do you securely send the request? How does your client register and obtain tokens?

So yes, the puck is going towards JWT... but that's like saying the puck is going to JSON/REST... If you are using OAuth2 to authenticate a person, and you are using JWT, then you will have to define a lot of details to do so securely. Instead of making up your own recipe (which you probably don't have the time to do...), leverage the best practices defined by the

experts at Google and Microsoft.

2 ^ v • Reply • Share ›

Clinton Blackburn ➚ Mike Schwartz • 2 years ago

I am in favor of using JWT access tokens, as opposed to OIDC, primarily due to the lack of support amongst Python/Django libraries. Given that we adopted OIDC as a mechanism for single sign-on, the primary information we need for creating a user in our micro-services are username and email. The basic contents of the JWTs we use closely follow the id_token spec (e.g. preferred_username and email values for profile data).

> How do you know that the JWT that you got back in the response relates to the request you sent?

Perhaps I am misunderstanding your question; but, when I make a request to a web server, it provides a response. The same applies to OAuth/OIDC requests. If you cannot trust an HTTPS connection, the Internet is fundamentally broken.

> How do you know that the access token has not been modified?

JWT encryption/signing solve this issue. If you, as a provider, support authentication via JWT, you should be validating the provided JWT.

> How do you know the code has not been modified?

I admit that at_hash and c_hash are new to me. We don't use them for our current implementation of OIDC. That said, my previous answer about JWT encryption/signing still applies.

> How do you securely send the request?

SSL/TLS. The OIDC spec doesn't dictate a transmission mechanism.

> How does your client register and obtain tokens?

Client management happens with OIDC the same way it happens with OAuth 2.0. In our case, we manually review/approve new client requests. Access tokens are issued

via the standard OAuth 2.0 flows.

1 ^ | v • Reply • Share ›

Mike Schwartz ➚ Clinton Blackburn • 2 years ago

Clinton,
TLS is required for OpenID Connect, as it should be for all API's. However, what we've seen is that TLS alone is not always enough. This is the reason that OpenID Connect Providers publish keys for signing and encryption. I think you are on board with this, because you also menitoned signinng your JWT.

Your answers are just too simplistic. You say "use JWT" without providing any of the details of how one would accomplish this. Great, you want to use signing and encryption--where do you publish the keys? How does the client register its public key? What algorthim should the client and OP use? If this were SAML, what you'd be saying is "use assertions", without providing any of the details of how to do so...

What you are describing is not using JWT... it is defining some domain specific mechanism for using JWT. What we've seen, is that is risky. Most developers don't build in the security protections necessary. They just "get it working" and move on--until their site gets hacked. Then they contribute to giving OAuth a bad name.

Also, regarding client registration... there is not much client metadata specified in OAuth 2.0. Check the IANA registrations, and what you'll see is that Connect defines more IANA OAuth parameters then OAuth itself.

https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml

For example, if you want to do private key authentication with your client--which has been a standard practice in enterprise for "web agents" for years... you're out of luck with plain OAuth.

Why is Connect not an IETF OAuth standard? There were politics--from what I hear it had something to do with the OAuth WG not wanting to take on authentication. In reality, Connect is a part of OAuth 2.0 at this point--it provides

the missing pieces that many people need to use OAuth for sign-in.

1 • Reply • Share ›

archenroot • 6 months ago

Oauth is authorization not authentication protocol, but OpenID connect basic profile is based on Oauth :D, I will die one day.

http://openid.net/specs/ope...

• Reply • Share ›

William Lowe Mod ➚ archenroot • 6 months ago

And to be technically correct, OpenID is *not* an authentication protocol either--it's an identity federation protocol. It standardizes how to securely share user information between autonomous domains (using OAuth 2.0).

FIDO U2F *is* an authentication protocol.

https://www.yubico.com/solu...

1 • Reply • Share ›

Tuttu • 7 months ago

Something great stuff !!!

• Reply • Share ›

Binh Thanh Nguyen • a year ago

Thanks, nice post

• Reply • Share ›

Paul Vixie • 2 years ago

how does this all relate to JWT, which is where many folks think the puck is going?

• Reply • Share ›

William Lowe Mod ➚ Paul Vixie • 2 years ago

See above, Paul. Thanks for the comment!

• Reply • Share ›

Schedule a call

Visit support