Token Authentication: The Secret to Scalable User Management

by Lindsay Brunner
May 11, 2016

General (https://stormpath.com/blog/category/general)

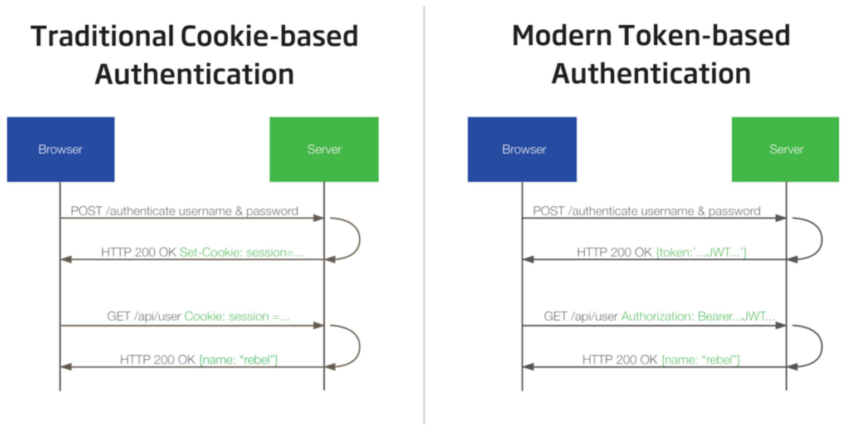
At Stormpath, we're in the business of authentication and authorization, which means we have lots of conversations with developers about user management, sessions, and scalability in web and mobile applications. We think token authentication (or token-based authentication) is one of the core elements of scalable identity and authorization management. Token authentication (https://stormpath.com/product/token-auth/) is stateless, secure, mobile-ready, and designed to grow with your user base without adding additional strain on your servers.

How Does Token Authentication Work?

Authentication (https://docs.stormpath.com/rest/product-guide/latest/auth_n.html#) is the process by which an application confirms user identity. Applications have traditionally persisted identity through session cookies, relying on session IDs stored server-side. This forces developers to create session storage that is either unique to each server, or implemented as a totally separate session storage layer.

Token authentication (https://docs.stormpath.com/rest/product-guide/latest/auth_n.html#how-token-based-authentication-works) is a more modern approach and is designed solve problems session IDs stored server-side can't. Using tokens in place of session IDs can lower your server load, streamline permission management, and provide better tools for supporting a distributed or cloud-based infrastructure.

In this method, tokens are generated for your users after they present verifiable credentials. The initial authentication could be by username/password credentials, API keys or even tokens from another service. (Stormpath's API Key Authentication Feature (http://docs.stormpath.com/guides/api-key-management/) is an example of this.)


Traditional Cookie-based Authentication
Browser
Server
POST /authenticate username & password
HTTP 200 OK Set-Cookie: session=...
GET /api/user Cookie: session =...
HTTP 200 OK {name: "rebel"}
Modern Token-based Authentication
Browser
Server
POST /authenticate username & password
HTTP 200 OK {token:'...JWT...'}
GET /api/user Authorization: Bearer...JWT...
HTTP 200 OK {name: "rebel"}

As you can see in the diagram above, once the user's credentials are exchanged for a token on the server, the client can use the token to validate each subsequent request. Once generated, the token is attached to the user via a browser cookie or saved in local/session storage (https://stormpath.com/blog/where-to-store-your-jwts-cookies-vs-html5-web-storage/).

The Token Convention

Let's try an analogy: When you attend a conference or convention, you're issued credentials specific to that event, often in the form of a lanyard. How did you score that lanyard? Well, you presented a valid, government-issued ID on the first day, they compared your ID, your face, and your registration. All matched, and the lanyard was yours. You've been authenticated. When you return to that convention the next day (a new session), or try to attend a gated sub-event (an authorization request), it's your event credentials that are checked, not your actual ID.

In a session ID world, each gatekeeper would need a list of every single attendee and their registrations (permissions!), and would check your valid, government-issued ID against that list not just each day, but every time you entered a new area or session. That sounds exhausting, right?

In token land, gatekeepers authenticate your identity and authorize your access based on the custom event credential you’re wearing around your neck. No additional ID needed, no enormous list to check every individual against. The lanyard carries all the information!

Why use Token Authentication?

1. Statelessness

Here’s the crucial bit for scalability: Your server will need to generate a token, but it will never need to store said token anywhere. All of the user metadata is encoded right into the token itself (https://stormpath.com/blog/token-auth-spa), so any machine on your network can validate any user. The server and client can pass the token back and forth forever and never store any user or session data. This is “statelessness,” and it’s the key to your application’s scalability.

2. Building a Mobile-Ready Backend

Using tokens for authentication in a mobile app (https://stormpath.com/blog/the-ultimate-guide-to-mobile-api-security) allow you to easily and securely control which mobile devices are accessing your API. Not only are they easier to use than cookies on iOS or Android, but they also allow your app to authenticate requests against multiple backends without extra effort on the part of your dev team.

3. Support for Multi-Server Platforms And Distributed Micro-Services

Applications supported by multiple and distributed servers will reap the greatest benefits from tokens, simply because of their stateless nature.

Similar to our convention analogy, imagine a user logs into your app via Server A. Their first several requests are also sent to Server A as it is perhaps nearest to them geographically. That's all well and good until a new request from that user triggers a shift to Server B, thanks to a load balancer or micro-service that's hosted from Server B.

With traditional session-based authentication, Server B would have to be set up to talk to the same distributed session storage layer as Server A, or you would have to set up "sticky sessions" on your load balancer (thus severely reducing the value of your load balancer). Yikes!

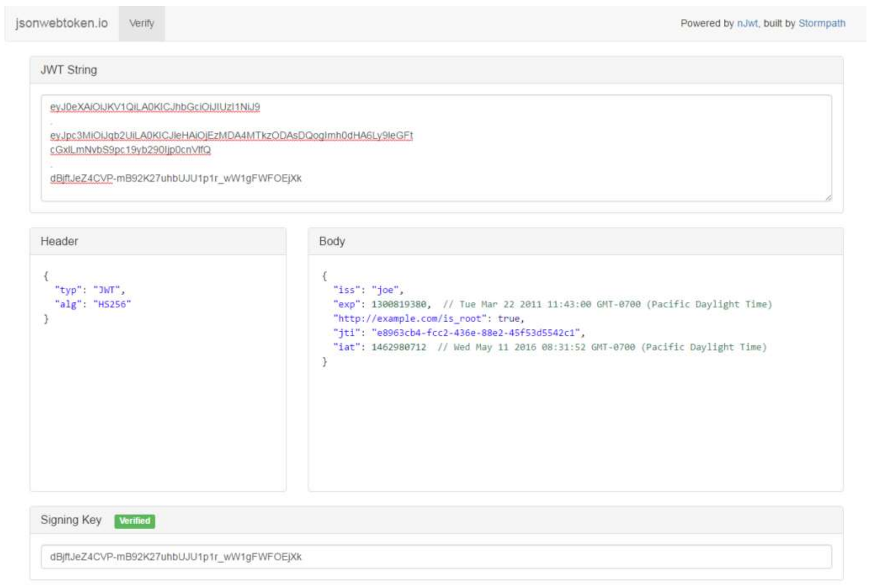
With a token, Server B already has everything it needs to know to validate the user's identity, no sticky sessions needed. Whew!

Anatomy of a Token

The JSON Web Token, or JWT, is the token solution that we recommend and use at Stormpath (https://stormpath.com/blog/jwt-the-right-way/). A JWT is a compact, URL-safe, encryptable JSON object that is rapidly becoming the standard for token implementation across the web. A JWT looks like any other ugly string but is separated into three sections by periods.

The first section, or header, describes the contents of the token. The second section, or payload (sometimes called "claims"), contains the identification data, authorization claims, and expiration time, as well as any custom data you choose to encode. The final section is the signature, a hash that cryptographically verifies the validity of the token.

This is a Base64 encoded string in Stormpath's own jsonwebtoken.io (https://www.jsonwebtoken.io):



```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
.
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

```
{
  "iss": "joe",
  "exp": 1300819380,  // Tue Mar 22 2011 11:43:00 GMT-0700 (Pacific Daylight Time)
  "http://example.com/is_root": true,
  "jti": "e8963cb4-fcc2-436e-88e2-45f53d5542c1",
  "iat": 1462980712  // Wed May 11 2016 08:31:52 GMT-0700 (Pacific Daylight Time)
}
```

```
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

(https://www.jsonwebtoken.io/)OAuth 2.0

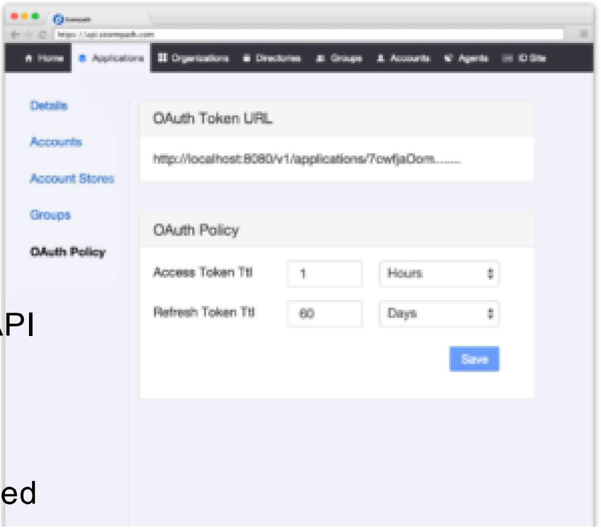
OAuth 2.0 is a framework that provides a set of protocols for interaction with a service that can delegate authentication or provide authorization. It is widely adopted across many mobile and web applications. Stormpath uses OAuth 2.0 because it is an industry standard that can be leveraged by any compliant library.

The Stormpath API currently support three of OAuth's various grant types:

- **Password Grant Type**: Provides the ability to get an Access Token based on a username and password.
- **Refresh Grant Type**: Provides the ability to generate another Access Token based on a special Refresh

Token.

- **Client Credentials Grant Type**: Provides the ability to exchange an API Key for an Access Token. This is supported through the API Key Management (http://docs.stormpath.com/guides/api-key-management/) feature.



OAuth Token URL

http://localhost:8080/v1/applications/7cwfjaOom......

OAuth Policy

Access Token Ttl 1 Hours

Refresh Token Ttl 60 Days

Types of Tokens

Within the OAuth 2.0 paradigm, there are two token types: Access and Refresh Tokens. Access Tokens grant access to a protected resource. Refresh Tokens are used to generate additional Access Tokens, without requiring the original credentials to be collected again. Both Access and Refresh Tokens have built-in security to prevent tampering and are only valid for a specific duration.

When you first authenticate, your application (and thus your user), is given both tokens, but the Access Token is set to expire after a short period (this duration is under your control, either through the Stormpath API or Admin Console). Once the initial Access Token has expired, the Refresh Token will allow your application to obtain a new Access Token. Refresh Tokens have a set expiration, allowing for unlimited use up until that expiration point is reached.

Stormpath and Token CRUD

As a developer, you can use Stormpath for full CRUD support, including the ability to issue and revoke Access and Refresh JWTs using OAuth 2.0 (https://stormpath.com/blog/token-auth-for-java) Password and Refresh Grant flows. Every Stormpath Application resource includes a built-in endpoint that supports these OAuth flows.

Once an Access Token has been generated, the token can be used to authorize individual requests made by your users as they are passed to your application. Your application will validate the token sent along with each request. In the event the Access Token has expired, your application can generate a new one based on the user's Refresh Token, without having to re-enter their original credentials.

It's worth noting that Stormpath does not automatically generate a new Refresh Token (or modify its expiration time). This means that once the Refresh Token expires, the user must re-authenticate to obtain new Access and Refresh Tokens.

These token lifetimes are configurable, so you can set the "time to live" (or TTL) both for your Access and Refresh Tokens. This is important to the security of your application because it gives you control over when your tokens expire. For security-critical applications, it's a common practice to have short TTLs for Access Tokens. This practice reduces vulnerability by forcing more frequent token refreshes against Stormpath.

Both Access and Refresh tokens can be revoked (independently, or together). For example, if a user's device is stolen or compromised, you can revoke their current tokens and force them to log in again.

Java and JJWT

JJWT (https://stormpath.com/blog/jjwt-how-it-works-why/) is a Java library providing end-to-end JWT creation and verification, developed by our very own Les Hazlewood. Forever free and open-source (Apache License, Version 2.0), JJWT is simple to use and understand. It was designed with a builder-focused fluent interface hiding most of its complexity. We'd love to have you try it out (https://github.com/jwtk/jjwt), and let us know what you think! (And, if you're a Node developer, check out NJWT (https://github.com/jwtk/njwt)!)

Stormpath Support for Token Authentication

With Stormpath, you can generate, manage, verify, and revoke OAuth tokens without any custom code. Check out these additional resources on user identity management (https://stormpath.com/product):

- Client-side token authentication for frameworks like Angular.js (https://stormpath.com/blog/jwt-authentication-angularjs/)
- Token authentication for web apps (https://stormpath.com/blog/token-auth-spa/)
- Social authentication (https://stormpath.com/product/social-login/) (Facebook, Google, etc.)
- Token authentication in PhP (https://stormpath.com/blog/token-management-in-php)
- OAuth 2.0 with Java Spring Boot (https://stormpath.com/blog/fun-with-java-spring-boot-token-management/)

Or, reach out and talk directly with one of our architects! Email us at support@stormpath.com (mailto:support@stormpath.com) to get started.

Search …

Explore the Topic

.NET (https://stormpath.com/blog/category/net)

General (https://stormpath.com/blog/category/general)

Java (https://stormpath.com/blog/category/java)

Javascript (https://stormpath.com/blog/category/javascript)

Mobile (https://stormpath.com/blog/category/mobile)

Node (https://stormpath.com/blog/category/node)

PHP (https://stormpath.com/blog/category/php)

Python (https://stormpath.com/blog/category/python)

REST API (https://stormpath.com/blog/category/rest-api)

Share a Post

5 0 0 0

Support: developers@okta.com