SON WEB TOKENS

- Json Benefits
- Json Architecture

OAUTH 2.0

- OAuth Frame-Work

Session ID

Cookies

Authentication

Who you are

Authorization

What you can do

I. JSON Web Tokens

JSON Web Token (JWT), pronounced "jot", is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. Because of its relatively small size, a JWT can be sent through a URL, through a POST parameter, or inside an HTTP header, and it is transmitted quickly. A JWT contains all the required information about an entity to avoid querying a database more than once. The recipient of a JWT also does not need to call a server to validate the token.

NOTE: JWT is used for Authorization Not Authentication

Benefits of JWT when compared to Simple Web Token (SWT) and Security Assertion Markup Language (SAML) tokens.

- More compact: JSON is less verbose than XML, so when it is encoded, a JWT is smaller than a SAML token. This makes JWT a good choice to be passed in HTML and HTTP environments.

- More secure: JWTs can use a public/private key pair in the form of an X.509 certificate for signing. A JWT can also be symmetrically signed by a shared secret using the HMAC algorithm. And while SAML tokens can use public/private key pairs like JWT, signing XML with XML Digital Signature without introducing obscure security holes is very difficult when compared to the simplicity of signing JSON. Read more about JWT signing algorithms in our blog.

- More common: JSON parsers are common in most programming languages because they map directly to objects. Conversely, XML doesn't have a natural document-to-object mapping. This makes it easier to work with JWT than SAML assertions.

- Easier to process: JWT is used at internet scale. This means that it is easier to process on user's devices, especially mobile.

JWT is a standard, which means that all JWTs are tokens, but not all tokens are JWTs. JWTs can be used in various ways:

- Authentication: When a user successfully logs in using their credentials, an ID Token is returned. According to the OpenID Connect (OIDC) specs, an ID Token is always a JWT.

- Authorization: Once a user is successfully logged in, an application may request to access routes, services, or resources (e.g., APIs) on behalf of that user. To do

so, in every request, it must pass an Access Token, which *may* be in the form of a JWT. Single Sign-on (SSO) widely uses JWT because of the small overhead of the format, and its ability to easily be used across different domains.

- Information Exchange: JWTs are a good way of securely transmitting information between parties because they can be signed, which means you can be sure that the senders are who they say they are. Additionally, the structure of a JWT allows you to verify that the content hasn't been tampered with.

Security

The information contained within the JSON object can be verified and trusted because it is digitally signed. Although JWTs can also be encrypted to provide secrecy between parties, Auth0-issued JWTs are JSON Web Signatures (JWS), meaning they are signed rather than encrypted. As such, we will focus on *signed* tokens, which can *verify the integrity* of the claims contained within them, while encrypted tokens *hide* those claims from other parties.

In general, JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA (although Auth0 supports only HMAC and RSA). When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

Before a received JWT is used, it should be properly validated using its signature. Note that a successfully validated token only means that the information contained within the token has not been modified by anyone else. This doesn't mean that others weren't able to see the content, which is stored in plain text. Because of this, you should never store sensitive information inside a JWT and should take other steps to ensure that JWTs are not intercepted, such as by sending JWTs only over HTTPS, following best practices, and using only secure and up-to-date libraries.

JWT ARCHITECTURE:

A well-formed JSON Web Token (JWT) consists of three concatenated Base64url-encoded strings, separated by dots (`.`):

- Header: contains metadata about the type of token and the cryptographic algorithms used to secure its contents.
- Payload (set of claims): contains verifiable security statements, such as the identity of the user and the permissions they are allowed.
- Signature: used to validate that the token is trustworthy and has not been tampered with. You must verify this signature before storing and using a JWT.

A JWT typically looks like this:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Header

The header *typically* consists of two parts: the hashing algorithm being used (e.g., HMAC SHA256 or RSA) and the type of the token (JWT).

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

Payload

The payload contains statements about the entity (typically, the user) and additional entity attributes, which are called claims. In this example, our entity is a user.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Signature

The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

To create the signature, the Base64-encoded header and payload are taken, along with a secret, and signed with the algorithm specified in the header.

For example, if you are creating a signature for a token using the HMAC SHA256 algorithm, you would do the following:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

II. OAuth 2.0

OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices. This specification and its extensions are being developed within the IETF OAuth Working Group.

Use OAuth access tokens if you want users to easily provide authorization to applications without needing to share private data or dig through developer documentation.

The OAuth 2.0 Authorization Framework

Abstract

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849.

III. What is a Session ID?

A *session ID* or *session token* is unique identifier that a website assigns to a specific user for some predetermined duration of time, or *session*, to keep track of visitor activity.

User authentication and shopping cart activity are two common uses for session IDs. The most common method of delivering and storing session IDs is with cookies; however, a session ID can also be embedded in a URL as a query string. Both Cookie and URL parameter session IDs can cause prevent search engines from properly crawling and indexing web content.

Session IDs Using Cookies

Session IDs stored in cookies are generally more secure than those transmitted through URL parameters.

Problem: Search engines ignore cookies.
Solution: Be sure to provide alternative methods for search engines to access web pages when cookies are used to present content, either through direct links or sitemaps.
When in doubt, *"Fetch as Googlebot"* from the site's associated Google Webmaster Toolsaccount, to see if cookies are preventing search bots from accessing any pages.

IV. Computer Cookie or HTTP Cookie

A computer “cookie” is more formally known as an HTTP cookie, a web cookie, an Internet cookie, or a browser cookie. The name is a shorter version of “magic cookie,” which is a term for a packet of data that a computer receives, then sends back without changing or altering it. No matter what it’s called, a computer cookie consists of information. When you visit a website, the website sends the cookie to your computer. Your computer stores it in a file located inside your web browser. (To help you find it, this file is often called “Cookies.”)

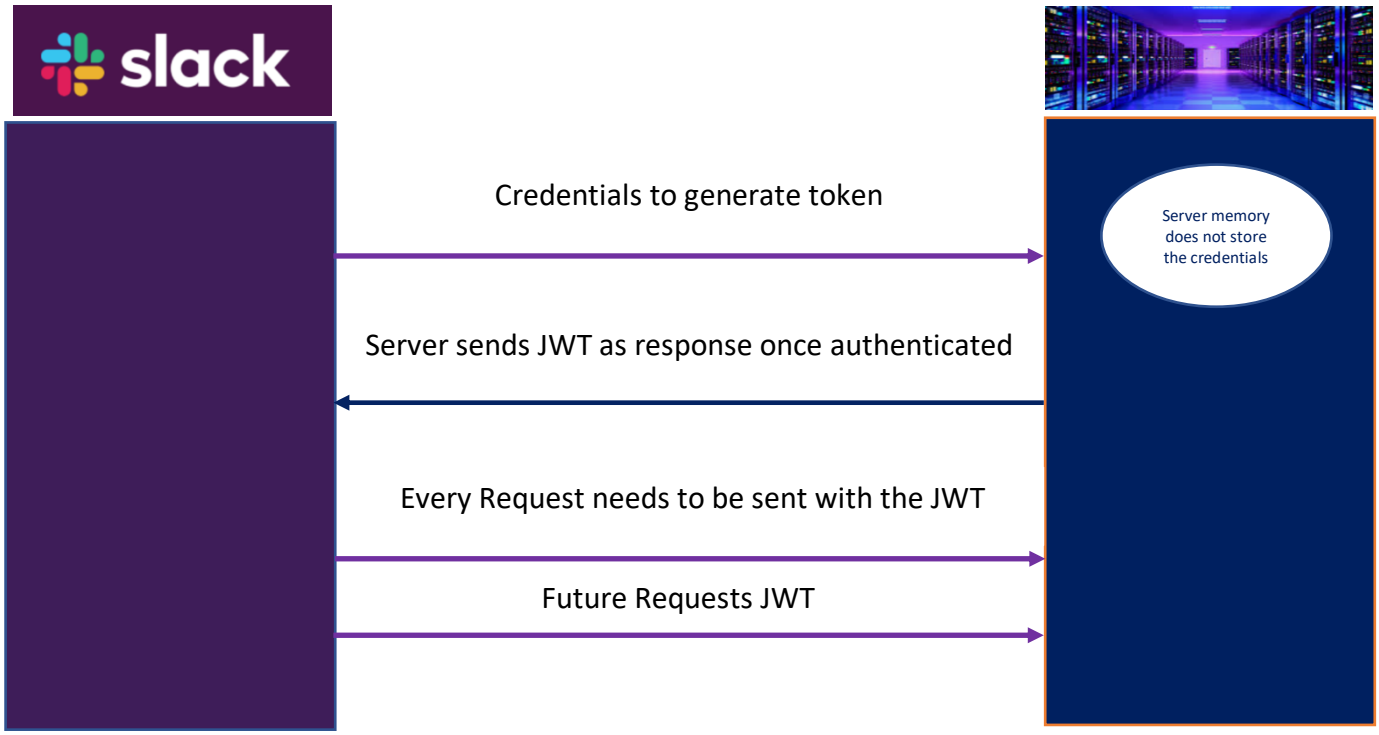
What Do Cookies Do?

The purpose of the cookie is to help the website keep track of your visits and activity. This isn’t always a bad thing. For example, many online retailers use cookies to keep track of the items in a user’s shopping cart as they explore the site. Without cookies, your shopping cart would reset to zero every time you clicked a new link on the site. That would make it impossible to buy anything online!

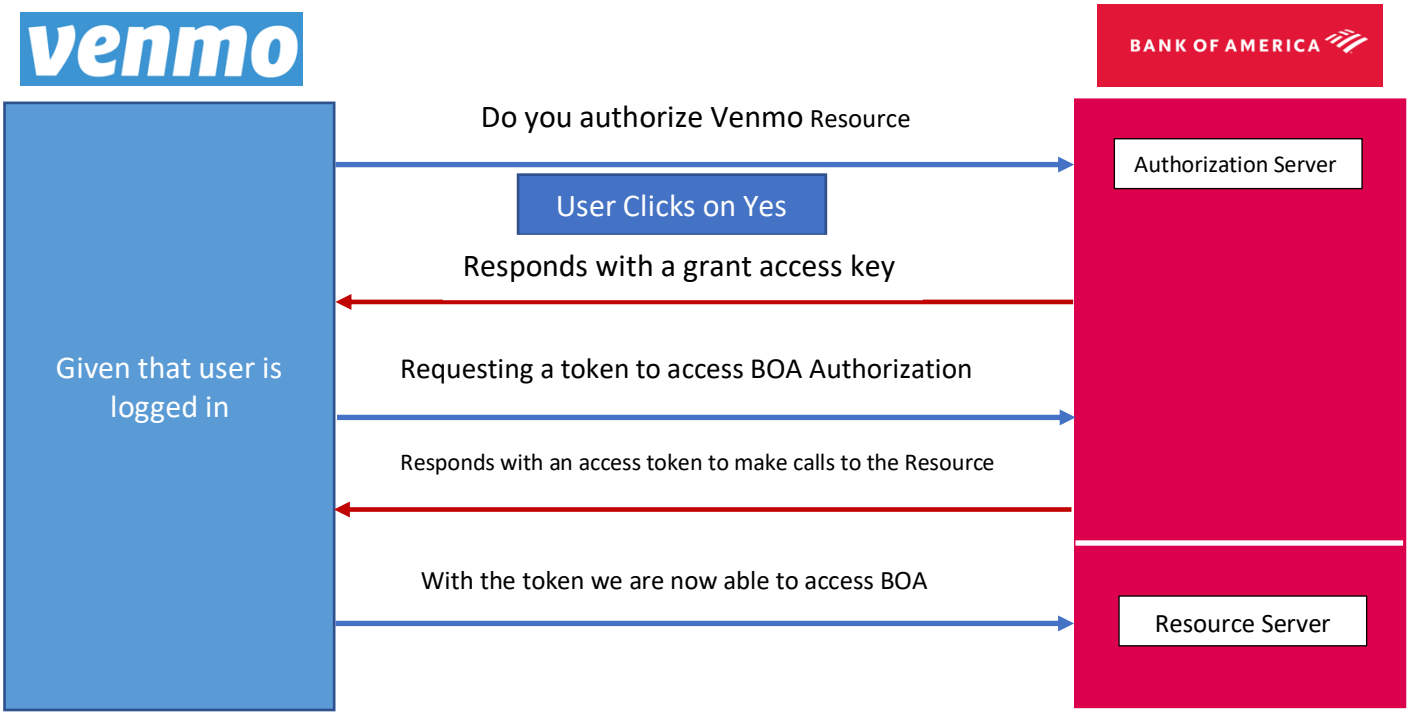
A website might also use cookies to keep a record of your most recent visit or to record your login information. Many people find this useful so that they can store passwords on commonly used sites, or simply so they know what they have visited or downloaded in the past.

Different types of cookies keep track of different activities. Session cookies are used only when a person is actively navigating a website; once you leave the site, the session cookie disappears. Tracking cookies may be used to create long-term records of multiple visits to the same site. Authentication cookies track whether a user is logged in, and if so, under what name.

Generating JSON Web Token



(OAuth 2.0 Architecture)



Cookies With Session ID