JSON Web Tokes

How to represent security claims between two parties?

Introduction

- **JSON Web Token (JWT)** is an open standard (RFC 7519) that defines a **compact** and **self-contained** way for **securely transmitting information** between parties as a JSON object.
- This information can be verified and trusted because it is **digitally signed** (secret or public/private key pair).
- We will focus on **signed tokes**. Signed tokens can verify the **integrity** of the claims contained within it.

When should you use JSON Web Tokens?

- **Authorization** (most common scenarion)

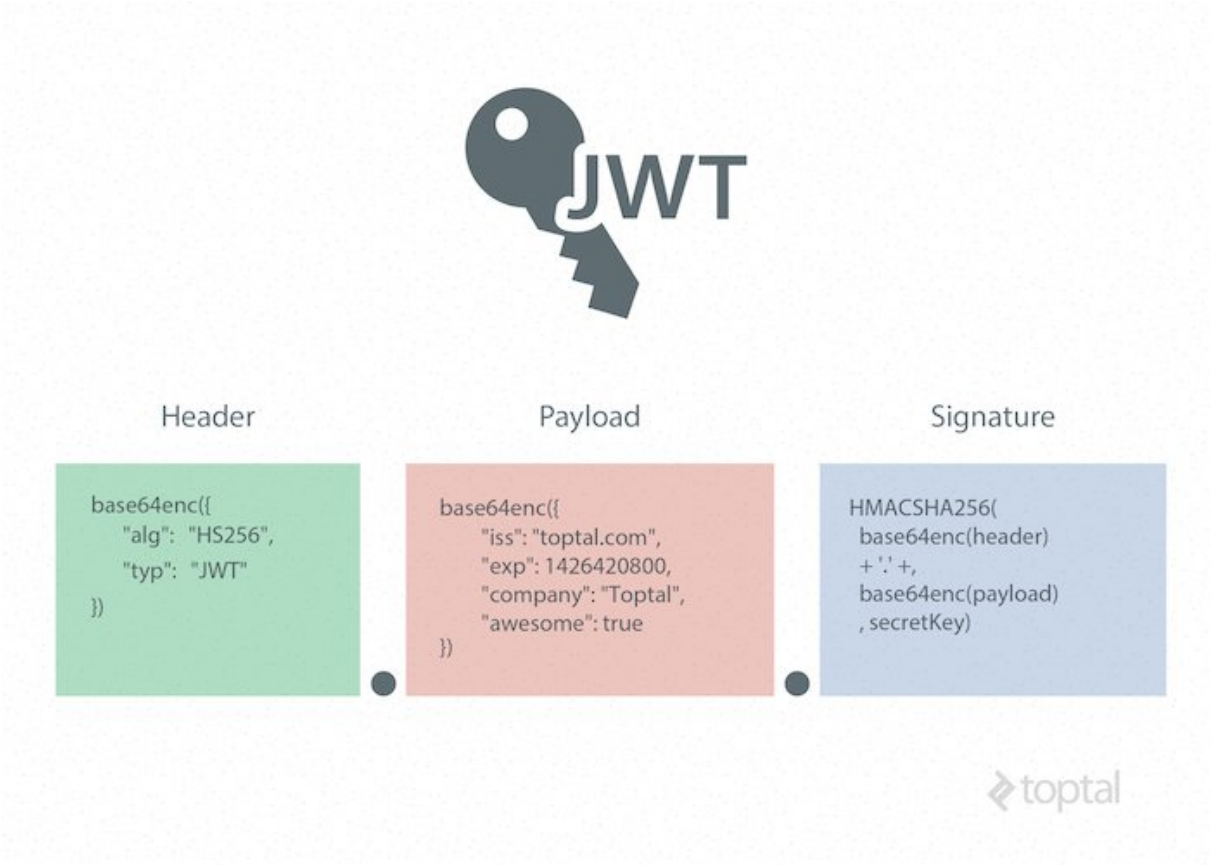
  Once user is logged in (*authenticated*), each subsequent request will include the JWT, allowing the user to access routes, services ... (*authorized*).

- **Information Exchange**

  Securely transmit information between two parties, because JWT can be signed.

What is the JSON Web Tokens structure?

- A JWT consists of three parts separated by dots (.):



Source: https://www.toptal.com/web/cookie-free-authentication-with-json-web-tokens-an-example-in-laravel-and-angularjs

---

Header

- Contains the metadata for the token and it minimally contains the type of signature and the encryption algorithm.

  Example:

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

- Once this is base64 encoded, we have the first part of our JWT:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

Payload

- A claim can be defined as a statement about an entity (typically, the user), as well as additional metadata about the token itself.
- Claims can be:
  - **Registered** (at IANA JSON Web Token Claims registry)
  - **Public** (have to be collision-resistant), e.g. `https://www.toptal.com/jwt_claims/is_admin`
  - **Private** (for closed environments)
- Example:

```
{
   "iss": "toptal.com",
   "exp": 1426420800,
   "https://www.toptal.com/jwt_claims/is_admin": true,
   "company": "Toptal",
   "awesome": true
}
```

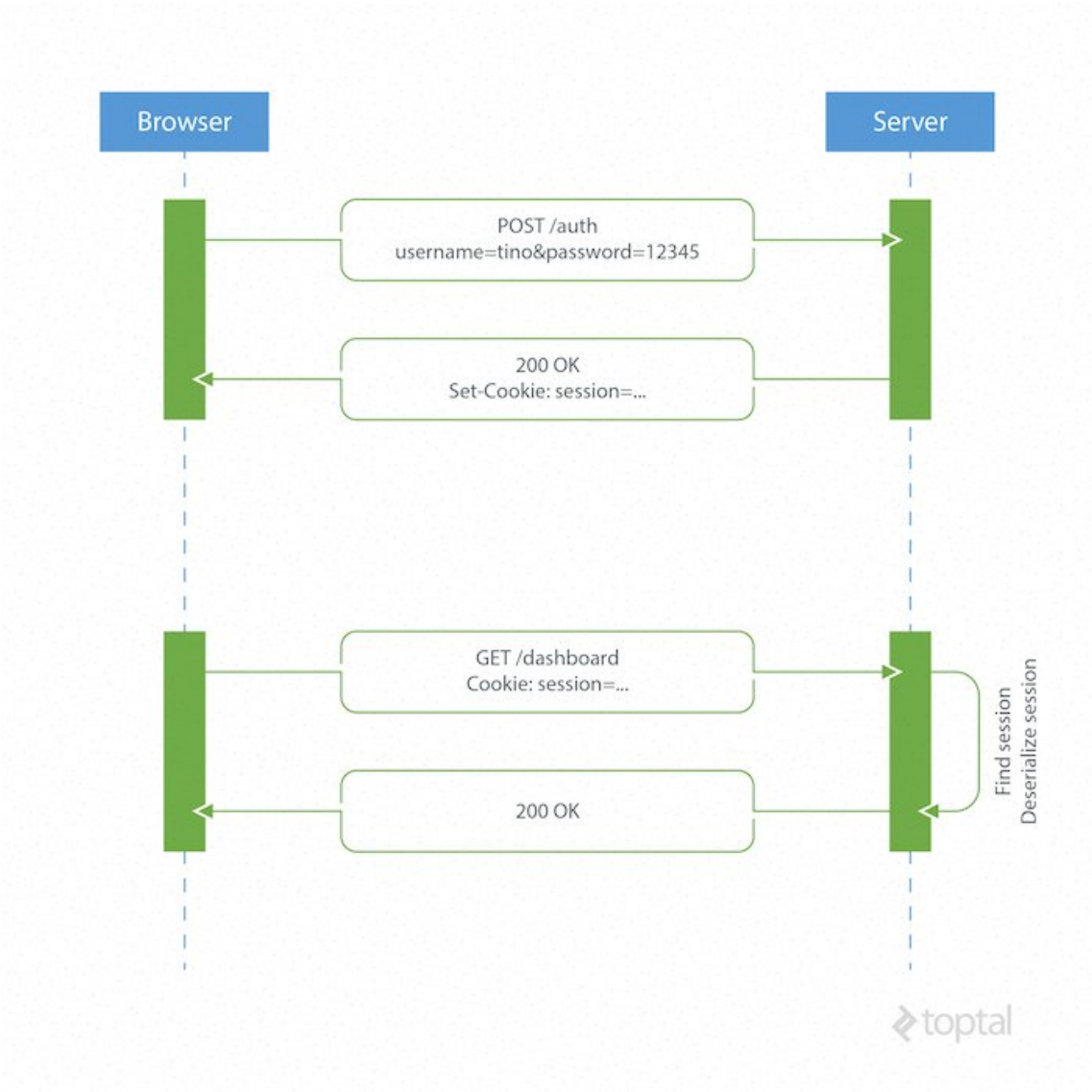
Signature

- The JWT standard follows the JSON Web Signature (JWS) specification (RFC7515) to generate the final signed token.
- It is generated by **combining** the **encoded JWT Header** and the **encoded JWT Payload**, and signing it using a strong encryption algorithm, such as HMAC SHA-256.
- The signature's secret key is held by the server so it will be able to verify existing tokens and sign new ones.

```
$encodedContent = base64UrlEncode(header) + "." + base64UrlEncode(payload);
$signature = hashHmacSHA256($encodedContent);
```

Why the need for Web Tokens?

- Traditional server based authentication:

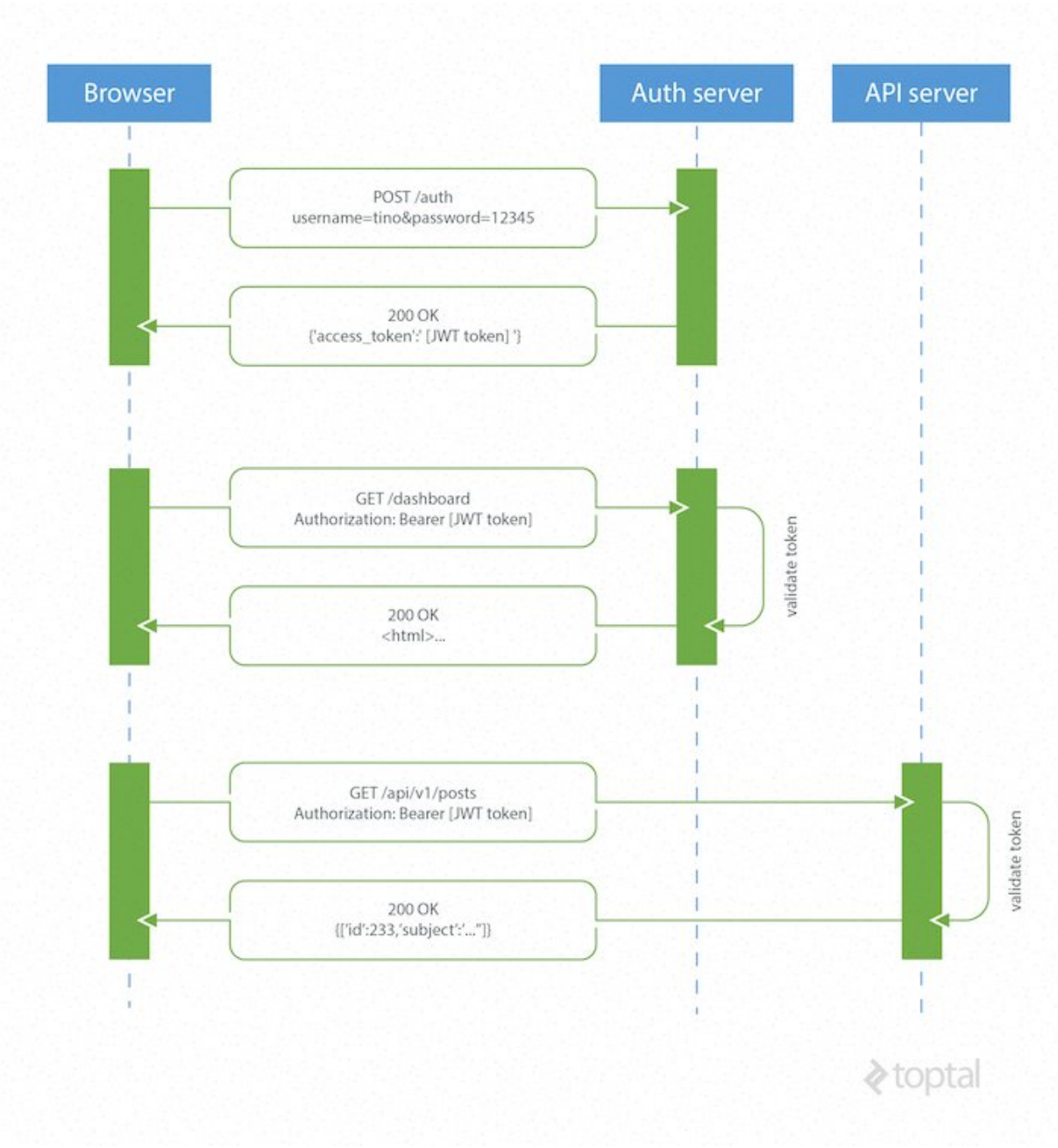


Drawbacks of Server-Based Authentication

- Because the HTTP protocol is stateless, there needs to be a mechanism for storing user information and a way to authenticate the user on every subsequent request after login.
- Most websites use cookies for storing user's session ID.

- On every subsequent request, the server needs to find that session and deserialize it, because user data is stored on the server.
- **Drawbacks**:
  - **Hard to scale:** The server needs to create a session for a user and persist it somewhere on the server. This can be done in memory or in a database. If we have a distributed system, we have to make sure that we use a separate session storage that is not coupled to the application server.
  - **Cross-origin request sharing (CORS):** When using AJAX calls to fetch a resource from another domain ("cross-origin") we could run into problems with forbidden requests because, by default, HTTP requests don't include cookies on cross-origin requests.
  - **Coupling with the web framework:** When using server-based authentication we are tied to our framework's authentication scheme. It is really hard, or even impossible, to share session data between different web frameworks written in different programming languages.

Token-Based Authentication



Advantages

- Token based/JWT authentication is stateless, so there is no need to store user information in the session. This gives us the ability to scale our application without worrying where the user has logged in. We can easily use the same token for fetching a secure resource from a domain other than the one we are logged in to.
- **Advantages:**
  - **Stateless, easier to scale:** The token contains all the information to identify the user, eliminating the need for the session state. If we use a load balancer, we can pass the user to any server, instead of being bound to the same server we logged in on.
  - **Reusability:** We can have many separate servers, running on multiple platforms and domains, reusing the same token for authenticating the user. It is easy to build an application that shares permissions with another application.
  - **JWT Security:** Since we are not using cookies, we don't have to protect against cross-site request forgery (CSRF) attacks. We should still encrypt our tokens using JWE if we have to put any sensitive information in them, and transmit our tokens over HTTPS to prevent man-in-the-middle attacks.
  - **Performance:** There is no server side lookup to find and deserialize the session on each request. The only thing we have to do is calculate the HMAC SHA-256 to validate the token and parse its content.

Further Readings

- JWT Introduction
- JSON Web Token Tutorial (**ATTENTION:** This tutorial uses the outdated AngularJS)