Rachit Gulati Seguir
Comprender cómo funciona la computadora. Amor a los humanos Experimentando Frontend. Trekker. Chico curioso. Noob Blogger.
13 de enero · 6 minutos de lectura

Parte 2: JWT para autenticar los API de los Servidores

¿Qué es JWT? (Resumen)

Según *openid*

> *JSON Web Token (JWT) es un medio para representar reclamos que se transfieren entre dos partes. Los reclamos en un JWT están codificados como un objeto JSON que está firmado digitalmente usando JSON Web Signature (JWS) y / o cifrado mediante JSON Web Encryption (JWE).*

En términos simples, es solo otra forma de codificar el objeto JSON y usar ese objeto codificado como tokens de acceso para la autenticación del servidor.

Esta es la segunda parte de la serie de dos publicaciones cortas sobre ***la aplicación práctica de JWT.***

1. JWT para descargar los archivos en el cliente.
2. JWT para la autenticación de servidor a servidor (publicación de blog actual).

Esta publicación de blog incluye los siguientes temas en detalle:

1. Partes de token JWT.
2. Cómo autenticar los API de los servidores (**concepto de productor y consumidor**).

Vamos a romper JWT ⛏

Partes de Token JWT

Generalmente hay tres partes en JWT como se muestra en la imagen de arriba.

1ra parte es HEADER:

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

Alg: Tenemos dos algoritmos principales (HS256 / RS256) para firmar nuestra tercera parte JWT (Firma) que mencionamos en los encabezados para que el productor y el consumidor (*lo comprendan pronto en la próxima sección*) ambos usen el mismo algoritmo para verificar el token en cada extremo. **HS256 indica que este token está firmado usando HMAC-SHA256.**

typ: Definir el tipo de token que es JWT obviamente en nuestro caso.

Cuando **base64UrlEncode** los datos de encabezado anteriores obtendremos *eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9* la primera parte de nuestro token JWT.

La segunda parte es PAYLOAD:

En su mayoría contiene los reclamos (datos personalizados) y algunos reclamos estándar también. Podemos usar reclamos estándar para identificar muchas cosas `exp: the expiry of the token` , es decir , `iat: time at which token issued` *etc.*

```
{
"email": "John Doe",
"xyz": "abc"
}
```

Cuando **base64UrlEncode** los datos de la carga útil anterior obtendremos *eyJuYW1lIjoiSm9obiBEb2UiLCJhbW91bnQiOjUwMCwieHl6IjoiYWJjIn0* la segunda parte de nuestro token JWT.

Las siguientes son algunas afirmaciones estándar: Wikipedia

> *Emisor (* `iss` *): identifica el principal que emitió el JWT;*

> *Subject (* `sub` *): identifica el sujeto del JWT;*

> *Audience (* `aud` *) - The "aud" (audience) claim identifies the recipients that the JWT is intended for. Each principal intended to process the JWT* ***must*** *identify itself with a value in the audience claim. If the principal processing the claim does not identify itself with a value in the* `aud` *claim when this claim is present, then the JWT* ***must*** *be rejected.*

> *Expiration time (* `exp` *) - The "exp" (expiration time) claim identifies the expiration time on or after which the JWT* ***must not*** *be accepted for processing. The value should be in NumericDate[10][11] format.*

> *Not before (* `nbf` *) - Similarly, the not-before time claim identifies the time on which the JWT will start to be accepted for processing.*

> *Issued at (* `iat` *) - The "iat" (issued at) claim identifies the time at which the JWT was issued.*

> *JWT ID (* `jti` *) - case sensitive unique identifier of the token even among different issuers.*

3rd part is SIGNATURE:

It is calculated by base64url encoding of header and payload and concatenating them with a period as a separator. Then encrypt it with HMAC-SHA256 along with the secret key and the result of the first step.

```
key           = 'secretkey';
unsignedToken = encodeBase64Url(header) + '.' +
encodeBase64Url(payload);
```

```
signature     = HMAC-SHA256(key, unsignedToken) // As
mentioned in header section.
```

When we perform the above steps we will get *54W-Y-Xz6xKgSnbQ7Se7tK5hcbXIvjsZ47u6CnQxjag* the third part of our JWT token

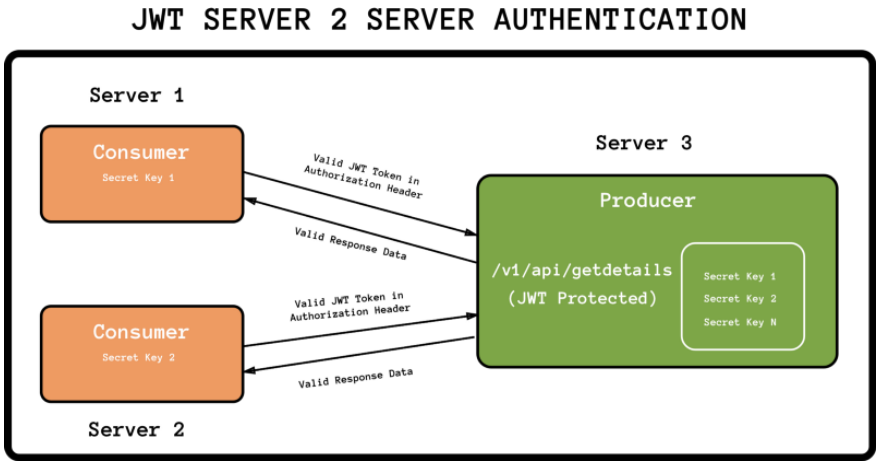
Do check out JWT.IO for the creation of JWT token online.

Producer and Consumer concept of API's 😎

There are two parties involved, one party who gives a service, and the other party who uses the service.

Producer is the one who gives a service. It will be the provider(Server) of the API(s) which are JWT protected.

Consumer is the one who uses it. It will be the customer(Server/Mobile App/ Web App/ Client) who will be providing the valid JWT token to consume the API(s) being provided by the Producer.



En la autenticación de servidor del servidor 2, ambas partes deben compartir el contrato personalizado para una API específica o para todas las API. Este contrato puede consistir en cualquier cláusula personalizada que desee presentar. Por ejemplo, aquí está el contrato personalizado a continuación que utilizaremos para nuestro ejemplo

1. **Comparta el SECRETO:** esta es la responsabilidad del lado Productor de compartir el secreto mutuo. Este secreto será necesario

para verificar el token en el extremo Productor y se usará el mismo secreto para crear el token en el lado del consumidor respectivo.

2. **Prepare el PAYLOAD: el** consumidor debe codificar todos los datos (cuerpo o consulta o params) en la carga útil del token JWT (puede elegir campos específicos que deben estar presentes en la carga útil de JWT pero le sugiero que envuelva todo el datos). Lo haremos al final del productor para verificar que los datos sean los mismos en la carga útil del token y en la API de solicitud.
es decir: la llamada GET a `/ v1 / api / getdetails? email = rachitgulati26 @ gmail.com` debe tener la carga JWT

```
{
   email: 'rachitgulati26@gmail.com'
}
```

y request.query también es el mismo que el anterior.

3. **OBTENER el TOKEN:** el token debe estar presente en el encabezado con el nombre ***jwt-token*** *(puede elegir su nombre personalizado o enviarlo en el encabezado de autorización después de todo su contrato personalizado)*. Además, la mejor práctica es enviarlo a través del esquema de Autorización del portador.

> *El mejor encabezado HTTP para que su cliente envíe un token de acceso (JWT o cualquier otro token) es el* `Authorization` *encabezado con el* `Bearer` *esquema de autenticación.*

4. **Idenitify the CONSUMER:** Solo necesitamos una última cosa y eso es identificar a nuestro consumidor. Esto podemos hacerlo de cualquier manera estableciendo `iss: CONSUMER_NAME` *un reclamo estándar en la carga útil o enviando otro encabezado* **jwt-consumer: CONSUMER_NAME.** Usaremos el último.

EMPLEO DEL CONSUMIDOR:

```
importar axios de 'axios'; // npm instala axios
```

```
const jwt = require ('jsonwebtoken'); // npm install
jsonwebtoken
const PRODUCER_URL = 'https: // <BASE_URL> / v1 / api /
getdetails';
```

```
/ ************ MANTENGALO SEGURO ****************** /
// Mantenlo en ENV y obténelo como process.env. ( secreto |
nombreCliente);
const secret = 'hello-reader';
const clientName = 'consumer-1-erx97812'
/ ************** END ***************** /
```

```
const getUserDetails = (email) => {
  const params = {
     email
   };
  // Firme todo aquí para ser cuerpo (POST, PUT, etc.) o
params (GET, POST, etc.).
  / *************** CREACIÓN DE
   TOKEN **************** / const token = jwt.sign ({...
params}, secreto) ;
   // BODY también: ex: jwt.sign ({... params, ... cuerpo},
secreto);
  / ***************** END ************** /
  const options = {
   params,
   encabezados: {
     'jwt-token': token, // Configuración de token en el
encabezado
     'jwt-consumer': clientName, // Consumer identity
   },
  };
  // De todos modos, te gusta llamar a una API externa.
Prefiero los axios.
  const response = axios.get (PRODUCER_URL, opciones);
  return response.data;
}
```

EMPLEOS DEL PRODUCTOR:

```
import _ from 'lodash'; // npm instala lodash
```

```
const router = express.Router ();
const jwt = require ('jsonwebtoken'); // npm instala
jsonwebtoken
```

```
export const API_HEADERS = {
  JWT_TOKEN: 'jwt-token',
  JWT_CONSUMER: 'jwt-consumer'
};
```

```
const MENSAJES = {
NOT_FOUND: 'Cabeceras válidas no presentes',
TOKEN_EXPIRES: 'Descarga token expiró',
USER_NOT_FOUND: 'Usuario con correo electrónico dado no',
NO_VALID_CLIENTE: 'No es un cliente válido',
};
/***************MISTERIOS***************/
```

```
// Preferiría mantener en la base de datos si es más de> 5.
Else // mantenerlo en el entorno.
```

```
const SECRETS = {
  'consumer-1-erx97812': 'secret1',
  'consumer-2-i32eecx2': 'secret2',
}
/ ************** END **** ********** /
```

```
// Middleware para JWT Verifier
```

```
export const JWTVerifier = async (req, res, next) => {
  const jwtToken = req.headers [API_HEADERS.JWT_TOKEN];
  const jwtConsumer = req.headers
[API_HEADERS.JWT_CONSUMER];
  const payload = {};
  if (! jwtToken ||! jwtConsumer) {
    return res.status (400) .json ({mensaje:
MESSAGES.NOT_FOUND});
  }
```

```
try {
    const secret = SECRETS [jwtConsumer];
    if (! secret) {
      return res.status (403) .json ({mensaje:
MESSAGES.NOT_VALID_CLIENT});
    }
    _.merge (carga útil, req.query, req.body);
    try {
      jwt.verify (bopClientToken, secret); // Verificar solo
token, no data.
      const decoded = jwt.decode (jwtToken, {complete:
true});
      // Verificando que los datos enviados dentro del token
deben ser iguales a la carga útil.
      if (! _. isEqual (decoded.payload, payload)) {
        return res.status (403) .json ({mensaje:
MESSAGES.NOT_VALID_PAYLOAD});
      }
      return next ();
    } catch (err) {
      return res.status (403) .json ({mensaje:
MESSAGES.NOT_FOUND});
    }
  } catch (error) {
    return next (error);
  }
};


router.get ('/ v1 / api / getdetails', JWTVerifier, (req,
res, next) {
  var datos personalizados = {
    ' abc@gmail.com ': {
      nombre: 'rachit',
      dob: '26 / 07/1993 '
    },
    ' xzy@gmail.com ': {
      name:' suchit ',
      dob: '09 /   09/1996 '
    }
  }
const user = customData [req.query.email];
  if (usuario) res.json ({ usuario});
```

```
    res.json ({mensaje: MESSAGES.USER_NOT_FOUND})
  })
```

Ahora entendemos qué es JWT y cómo podemos usarlo para autenticar nuestra comunicación Servidor-Servidor 👩. Gracias a nuestro **<JWT>.**

¿Quieres consultar la primera publicación del blog de esta serie? .

Por favor, siéntete libre de twittear @squiroid con cualquier pregunta, comentario, etc. :)

Si te gusta mi publicación y quieres que escriba más, haz clic en 👋 para darme una apreciación y amor ❤.