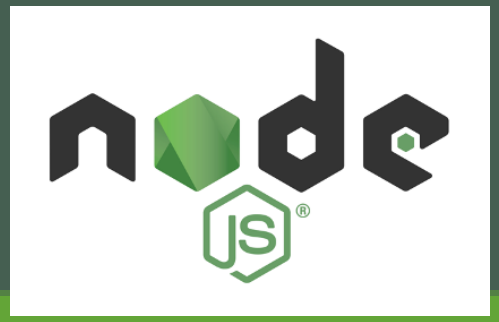


Web Authentication

Contenido

- HTTP Basic y Digest authentication
- Cookies
- Tokens
- Signatures
- One-time passwords
- JWT
- Passport.js

HTTP Basic Authentication: Cliente

- La autenticación HTTP Básica es un método que permite al cliente enviar un usuario y una password al hacer una petición
- Es la manera más simple de forzar el control de acceso y no requiere cookies o sessions
- Se aconseja usarlo con HTTPS
- Para usarlo, el cliente tiene que enviar una cabecera del mensaje con la propiedad **Authorization** junto a cada petición. **El usuario y la password no se encripta**, pero se construye de la siguiente manera:
  - se concatenan username y password en un string con la siguiente estructura:

    username:password
  - El string se codifica con Base64
  - Se coloca la palabra **Basic** delante del valor codificado:
  - Ej:

```
Authorization: Basic am9objoxMjM0
```

HTTP Basic Authentication: Servidor

- Express ya no implementa específicamente un módulo de autenticación.
- Aconseja utilizar: https://www.npmjs.com/package/basic-auth
- Ej:

```
//config/auth.js

const auth = require('basic-auth');
const admins = { 'john': { password: '1234' }, };

module.exports =
function (request, response, next)
{
var user = auth(request);
if (!user || !admins[user.name] ||
admins[user.name].password !== user.pass) {
response.set('WWW-Authenticate', 'Basic realm="example"');
return response.sendStatus(401);
}
return next();
};
```

```
//server.js

const express = require('express');
const auth = require('./config/auth')
const app = express();


app.use(auth)

app.get('/',(req,resp)=>{
resp.send(`Welcome user`)
})

app.listen('3000');
```

HTTPS: HTTP Secure

- Funciona desde el puerto 443 y utiliza un cifrado basado en SSL/TLS con el fin de crear un canal cifrado entre el cliente y el servidor.
- Se utilizan los protocolos SSL (Secure Sockets Layer) -deprecado- y TLS (Transmission Layer Security) para enviar paquetes cifrados a través de Internet. Se pueden utilizar para más de un protocolo, no sólo con HTTP. **HTTP + SSL/TLS = HTTPS**
- Se basa en el sistema clave pública-clave privada:
  - El administrador de un servidor Web crea un certificado de clave pública, firmado por una autoridad de certificación.
  - El cliente envía una petición cifrada con la clave pública que se descifra en el servidor con la clave privada.

HTTPS con Express

- Ej:
  - Instalar en Windows la aplicación openssl desde: https://slproweb.com/products/Win32OpenSSL.html
  - Crear un certificado autofirmado: `openssl req -nodes -new -x509 -keyout server.key -out server.cert`
  - Incorporar el cifrado al servidor:

```
//server.js

const express = require('express');
const auth = require('./config/auth')
const fs = require('fs')
const https = require('https')
const app = express();

app.use(auth)
app.get('/',(req,resp)=>{
resp.send(`Welcome user`)
})
https.createServer({
key: fs.readFileSync('server.key'),
cert: fs.readFileSync('server.cert')
}, app)
.listen(3000, function () {
console.log('Example app listening on port 3000! Go to https://localhost:3000/') })
```

Ejercicio 27

- Implementar la autenticación básica en el ejemplo 26.
  - Almacenar una password enviada por un usuario (parámetros del request) en la colección Persona en el campo password **codificada** en base64. (*tip: buscar codificación vía buffer en Node.js*)
  - Construir un módulo de autenticación en config/...
  - Proteger con autenticación el método get.

HTTP Basic Auth

- Desventajas:
  - El usuario y la password se envían con cada petición, de esta manera están expuestos incluso en servidores HTTPS
  - No hay manera de desconectar a un usuario que utiliza Basic auth
  - La expiración de las credenciales no es trivial. Se debe pedir al usuario que cambie la password

- Para obtener un certificado TLS gratis: https://letsencrypt.org/about/

HTTP Autenticación Digest

- Método que utiliza el password y otros bits de información para crear un hash que se envía al servidor para realizar la identificación

HTTP Digest Authentication	HTTP Basic Authentication
Las credenciales siempre viajan encriptadas	Credenciales en texto plano
No requiere otros sistemas de seguridad	Requiere otros sistemas de seguridad adicionales como HTTPS
Las credenciales deberían ser encriptadas, incluyendo username y realm	Las passwords pueden ser mantenidas encriptadas
Se puede configurar para mantener la integridad de los datos entrantes	Sólo tiene el propósito de la autenticación
Cada llamada requiere dos peticiones al cliente	Requiere SSL para ser más seguro, lo cual hace que sea más lento que HTTP

HTTP Autenticación Digest

- Pasos de la autenticación:
  - El cliente hace una petición inicial sin credenciales.
  - El servidor responde con un ***nonce***, que es un único string de datos generados por el servidor + un **realm** (que indica qué parte del site es accessible y qué tipo de credenciales debe proporcionar el cliente
  - El cliente responde con el *nonce* y las credenciales encriptadas – username, password y realm
  - El servidor sirve la información demandada o lanza un error
- Para incorporar autenticación HTTP Digest se pueden utilizar módulos predefinidos como http-digest

HTTP Autenticación Digest

- Ej:

```
// HTTP module
const http = require('http');

// Authentication module.
const auth = require('http-auth');
let digest = auth.digest({
    realm: "Simon Area.",
    file: __dirname + "/./data/users.htdigest"
// vivi:anna, sona:testpass
});
```

```
// Creating new HTTP server.
http.createServer(digest, (req, res) => {
    res.end(`Welcome to private area -
${req.user}!`);
}).listen(1337, () => {
    // Log URL.
    console.log("Server running at
http://127.0.0.1:1337/");
});
```

```
//data/users.htdigest
vivi:Simon Area.:fd805820dae2cf2672473464eaa4b414
sona:Simon Area.:86482b02439333102d7c6f374fc43afe
frida:Other Area.:8310c3dcea3a63dd0c76970d682abf0f
```

Tokens

- Porción de datos(texto) que identifica una petición. Normalmente se envía en la cabecera de la petición como cualquier dato de autenticación, como **session** y **cookies** (en el caso de HTTP)
- Ej de **token:**

```
Bearer 0a4d55a8d778e5022fab701977c5d840bbc486d0
```

- El token no contiene ninguna información, toda la información debe almacenarse en el servidor.
- Dos tipos de tokens:
  - tokens autocontenidos
  - tokens opacos

Tokens opacos (tokens de acceso)

- Son tokens que el cliente no puede ver, no requieren ser firmados ni divulgados sus protocolos
- No contiene información útil que pueda ser extraída. Funcionan de forma similar a ids de session, son sólo secuencias de caracteres.
- Los campos con información adicional deben almacenarse en ambas partes para establecer la comunicación.
- Se utilizan para informar a las API que el usuario ya ha sido autenticado por otros medios y puede acceder a la información.

Tokens transparentes o autocontenidos

- Son los más utilizados en la autenticación de APIs
- Permiten hacer aplicaciones escalables que puedan ser utilizadas con cualquier front-end
- El Cliente envía un código al servidor y el servidor se encarga de descifrarlo y comprobar la identidad del usuario, si está registrado y qué tipo de acceso tiene.
- El estándar más utilizado es JWT (JSON Web Token)
- Ej https://jwt.io/:


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJ
SMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

JWT (JSON WEB TOKEN)

- Tres partes:
  - **Header** (cabecera) tiene dos campos:
    - El tipo de token: **JWT**
    - El algoritmo utilizado para la encriptación: **HS256** o **RS256.**
      - **HS256:** Es un algoritmo simétrico, sólo requiere una clave secreta para desencriptarlo.
      - **RS256:** Es asimétrico, requiere clave pública y privada.

HEADER: ALGORITHM & TOKEN TYPE

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

JWT

- Tres partes:
  - **Payload** (Datos del usuario). Parte principal, presenta tres tipos de propiedades:
    - **Públicas**
    - **Privadas**
    - **Registradas**

PAYLOAD: DATA

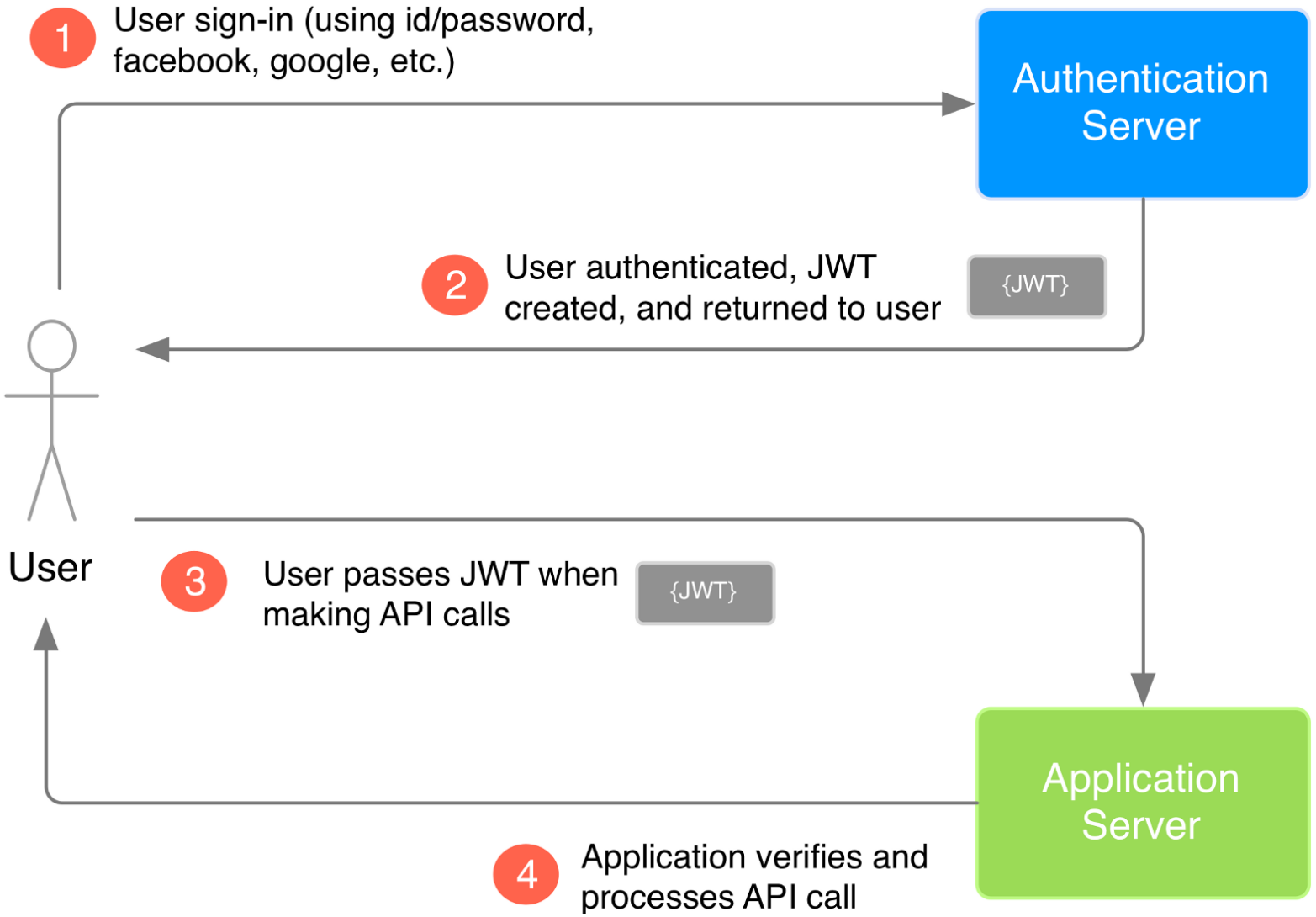

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```


http://www.iana.org/assignments/jwt/jwt.xhtml

JWT

- Tres partes:
  - **Signature:** La firma es la tercera y última parte del JSON Web Token. Está formada por los anteriores componentes (Header y Payload) cifrados en *Base64* con una clave secreta (almacenada en el servidor). Sirve de *Hash* para comprobar que todo está bien.

JWT con Node.JS



JWT con Node.JS

1. El usuario se registra
   - Se registran los datos con la password encriptada
2. El servidor genera un token basado en los datos del usuario registrado y lo envía al cliente
3. El cliente pasa el token cada vez que realiza una petición a la API
4. El servidor verifica el token y procesa la petición

JWT con Node.JS

1. El usuario se registra
   - Creamos un middleware de mongoose que procesa la encriptación de la clave con **bcrypt:**

```
UserSchema.pre('save', (next) => {
let user = this
//if (!user.isModified('password')) return next()

bcrypt.genSalt(10, (err, salt) => {
if (err) return next(err)

bcrypt.hash(user.password, salt, null, (err, hash) =>
{
if (err) return next(err)

user.password = hash
next()
})})})
```

JWT con Node.JS

2. Se crea un token de autenticación

```
const jwt = require('jwt-simple')
const moment = require('moment')
const config = require('../config')

function createToken (user) {
const payload = {
sub: user._id,
iat: moment().unix(),
exp: moment().add(14, 'days').unix()
}

return jwt.encode(payload, config.SECRET_TOKEN)
}
```

Se envía el token al cliente

```
req.user = user
res.status(200).send({
message: 'Te has logueado correctamente',
token: service.createToken(user)
})
```

JWT con Node.JS

4. El servidor verifica el token y procesa la petición

```
function isAuth (req, res, next) {
if (!req.headers.authorization) {
return res.status(403).send({ message: 'No tienes
autorización' })
}

const token = req.headers.authorization.split(' ')[1]

services.decodeToken(token)
.then(response => {
req.user = response
next()
})
.catch(response => {
res.status(response.status)
})
}
```

```
function decodeToken (token) {
const decoded = new Promise((resolve, reject) => {
try {
const payload = jwt.decode(token, config.SECRET_TOKEN)

if (payload.exp <= moment().unix()) {
reject({
status: 401,
message: 'El token ha expirado'
})
}
resolve(payload.sub)
} catch (err) {
reject({
status: 500,
message: 'Invalid Token'
}) } })
return decoded }
```

Autenticación con Passport

- Middleware de autenticación para NodeJS: `$ npm install passport`
- Los mecanismos de autenticación se denominan estrategias, y se instalan como módulos individuales. Ej: `$ npm install passport-local`

- Para autenticar se invoca el método Passport.authenticate() especificando la estrategia a emplear:

```
app.post('/login', passport.authenticate('local'), function(req, res) {
// Sí la función llamada tuvo éxito
// `req.user` contiene el usuario autenticado.
res.redirect('/users/' + req.user.username); });
```

- El proceso de autenticación puede llevar asociada una redirección de la respuesta:

```
app.post('/login', passport.authenticate('local',
{ successRedirect: '/',
  failureRedirect: '/login' }));
```

Configuración de la autenticación con Passport

- Estrategia de autenticación
  - Incluyen verificación de user y password, autenticación delegada vía OAuth o autenticación federada usando OpenID
  - Requieren ser configuradas mediante la función use()
  - Ej de configuración de validación user/password:

```
passport.use(new LocalStrategy(
 function(username, password, done) {
  User.findOne({ username: username }, function (err, user) {
    if (err) { return done(err); }
    if (!user) {
       return done(null, false, { message: 'Incorrect username.' });
    }
    if (!user.validPassword(password)) {
       return done(null, false, { message: 'Incorrect password.' });
    }
    return done(null, user);
   });
 }
));
```

Configuración de la autenticación con Passport

- Callback de verificación
  - Las estrategias requieren un callback de verificación de las credenciales de un usuario.
  - Cuando Passport autentica una petición, parsea las credenciales que contiene. Invoca al callback de verificación con las credenciales como argumentos. Si las credenciales son válidas, el callback invoca al método **done** pasándole el usuario autenticado

```
...
if (err) { return done(err); }
if (!user) {
        return done(null, false, { message: 'Incorrect username.' });
}
if (!user.validPassword(password)) {
        return done(null, false, { message: 'Incorrect password.' });
}
    return done(null, user);
...
```

Configuración de la autenticación con Passport

- Middleware
  - Se requiere inicializar Passport mediante el middleware **passport.initialized()**
  - En caso de usar sesiones, también se requiere utilizar **passport.session()**

```
var session = require("express-session"),
    bodyParser = require("body-parser");


app.use(express.static("public"));
app.use(session({ secret: "cats" }));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(passport.initialize());
app.use(passport.session());
```

Autenticación de APIs con Passport

- Lista de estrategias utilizadas en protección de *endpoints* de APIs:

Scheme	Specification	Developer
Anonymous	N/A	Jared Hanson
Bearer	RFC 6750	Jared Hanson
Basic	RFC 2617	Jared Hanson
Digest	RFC 2617	Jared Hanson
Hash	N/A	Yuri Karadzhov
Hawk	hueniverse/hawk	José F. Romaniello
Local API Key	N/A	Sudhakar Mani
OAuth	RFC 5849	Jared Hanson
OAuth 2.0 Client Password	RFC 6749	Jared Hanson
OAuth 2.0 JWT Client Assertion	draft-jones-oauth-jwt-bearer	xTuple
OAuth 2.0 Public Client	RFC 6749	Tim Shadel

OAuth 2.0 JWT

- Instalación estrategia oauth2-jwt-bearer: `$ npm install passport-oauth2-jwt-bearer`
- Configuración de la estrategia:

```
var ClientJWTBearerStrategy = require('passport-oauth2-jwt-bearer').Strategy;


passport.use(new ClientJWTBearerStrategy(
       function(claimSetIss, done) {
           Clients.findOne({ clientId: claimSetIss }, function (err, client) {
                     if (err) { return done(err); }
                     if (!client) { return done(null, false); } return done(null, client);
           });
        }
));
```

- Autenticación de la petición:

```
app.get('/profile',
passport.authenticate(['oauth2-jwt-bearer'], { session: false }),
oauth2orize.token());
```

Autenticación con Passport

- Autenticación con Facebook