Práctica 4: Seguridad en el desarrollo web

Calidad del Software 2017-2018

José Correro Barquín
Santiago Godoy Poce

7 de julio de 2018

Nivel A

OAuth 2.0

Para implementar este protocolo, se ha usado el middleware de autentificación Passport para aplicaciones Node.js que usen Express. En particular, se ha utilizado la estrategia de Passport para la autentificación con Google usando el API OAuth 2.0.

En primer lugar, hemos instalado el módulo con:

```
$ npm install passport-google-oauth20
```

Luego, se ha tenido que crear una aplicación de Google desde la Consola de Desarrolladores de Google para generar las credenciales necesarias para aplicar la estrategia. Además, se ha añadido una URL de redirección, que será la misma que pongamos en el parámetro callbackURL de la estrategia.

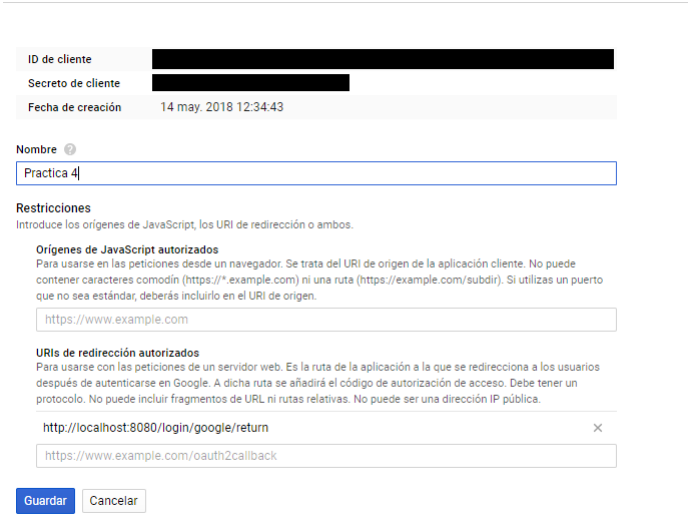

Figura 1: Consola de Desarrolladores de Google

A continuación, se implementa la estrategia en sí de la siguiente forma (se han obviado las credenciales):

```
passport.use(new GoogleStrategy({
    clientID: "",
    clientSecret: "",
    callbackURL: "http://localhost:8080/login/google/return",
    scope: "https://www.googleapis.com/auth/plus.login"
  },
  function(accessToken, refreshToken, profile, cb) {
    usersList.findByToken(accessToken, function (err, user) {
      return cb(err, user);
    });
  }
));
```

Figura 2: Implementación de la estrategia

Hemos usado un método dentro de la estrategia que hemos definido en *users-list* llamado *findByToken* que simplemente compara si el token de algún usuario almacenado en la 'base de datos' coincide con el del usuario que está tratando de acceder, si no existe, se crea un usuario de la siguiente forma:

```
exports.findByToken = function(token, cb) {
    process.nextTick(function () {
        let found = false;
        let user;
        for (let i = 0, len = users.length; i < len; i++) {
            user = users[i];
            if (user.token == token) {
                found = true;
                break;
            }
        }
        if(!found) {
            user = {
                id: users.length + 1,
                name: "Real Madrid",
                email: "halamadrid@gmail.com",
                password: "YNadaMas",
                token: token
            };
        }

        return cb(null, user);
```

Figura 3: Implementación de findByToken

A la estrategia también se le ha pasado un parámetro *scope*, con el ámbito, ya que si no, se produce un error *missing scope parameter*.
Una vez que se ha implementado la estrategia, es necesario añadir un par de opciones:

```
app.use(passport.initialize());
```

Que nos sirve para inicializar el módulo de autentificación.

```
app.use(passport.session());
```

Que utilizamos para cambiar el valor del usuario que está en las cookies del cliente por el objeto usuario deserializado.
Para realizar esto que hemos comentado, es necesaria la inclusión de dos métodos para serializar y deserializar el usuario, que son:

```
passport.serializeUser(function(user, done) {
done(null, user);
});

passport.deserializeUser(function(user, done) {
done(null, user);
});
```

Por último, se establecen las rutas para la URL, como la de *home*, la de *login*, la de *callback* o la del *perfil*.

```
app.get('/',
  function(req, res) {
    res.render('home', { user: req.user });
});

app.get('/login',
  function(req, res){
    res.render('login');
  });

app.get('/login/google',
  passport.authenticate('google', { scope: ['profile'] }));

app.get('/login/google/return',
  passport.authenticate('google', { failureRedirect: '/login' }),
  function(req, res) {
    // Successful authentication, redirect home.
    res.redirect('/');
  });

app.get('/profile',
  require('connect-ensure-login').ensureLoggedIn(),
  function(req, res){
    res.render('profile', { user: req.user });
  });
```

Figura 4: Definición de rutas

Se utiliza:

```
passport.authenticate()
```

Junto a la estrategia de 'google' para autentificar las peticiones.

Encriptación

Para la encriptación de datos se ha hecho uso de la librería CryptoJS. En este caso, se hace uso de codificación UTF-8 y Base64 y de una clave (constante) para encriptar tanto el mensaje (en este caso, el usuario) como una cabecera que se le añade al mensaje con el nombre de la asignatura.

```
const CryptoJS = require("crypto-js");

function toWordArray(str) {
    return CryptoJS.enc.Utf8.parse(str);
}

function toBase64String(words) {
    return CryptoJS.enc.Base64.stringify(words);
}

exports.encrypt = function(user) {
    var key = "ZxlNEnojO5HbQngiYvrqu32Br6V";
    var PROTOCOL_AES256 = 2;
    var secret_key = CryptoJS.SHA256(key);

    var json_payload = JSON.stringify(user);

    // Se crea la cabecera
    var header = toWordArray("CALIDADSOFTWARE" + String.fromCharCode(PROTOCOL_AES256));
    var iv = CryptoJS.lib.WordArray.random(16);
    var body = CryptoJS.AES.encrypt(json_payload, secret_key, { iv: iv });

    // Se añade la cabecera al mensaje
    header.concat(iv);
    header.concat(body.ciphertext);

    return toBase64String(header);
}
```

Figura 5: Implementación de CryptoJS