**Informe Técnico**

**Implementación de Autenticación Segura y Control de Acceso en Aplicaciones Web utilizando JWT y Bcrypt – Lección 4**

**1. Escenario**

Se desarrolló una aplicación de **gestión de proyectos** que debe contar con un sistema seguro de autenticación y control de acceso. Se definieron tres perfiles de usuario:

* **Administrador:** CRUD completo sobre proyectos.
* **Editor:** Ver, crear y editar proyectos, sin permisos de eliminación.
* **Usuario común:** Solo visualizar proyectos asignados.

El objetivo fue implementar un sistema completo de **autenticación con JWT** y **control de acceso por roles**, asegurando que cada usuario tenga acceso únicamente a las funciones que le corresponden.

**2. Metodología**

Se aplicaron buenas prácticas basadas en **OWASP Top 10** y estándares de seguridad:

1. **Registro seguro:** contraseñas almacenadas mediante **hash con bcrypt**.
2. **Inicio de sesión:** validación de credenciales y emisión de JWT con expiración.
3. **Middleware de autenticación:** validación de tokens antes de acceder a rutas protegidas.
4. **Control de acceso por roles:** funciones que restringen operaciones críticas a Administradores o Editores.
5. **Protección de API:** solo usuarios autenticados pueden interactuar con recursos sensibles.
6. **Cierre de sesión:** invalidación del JWT en cliente (eliminación de token en cookies o localStorage).

**3. Implementación Técnica**

**3.1 Registro de usuario (bcrypt + rol)**

const bcrypt = require('bcrypt');

const saltRounds = 12;

async function registrarUsuario(nombre, contraseña, rol = 'usuario') {

const hash = await bcrypt.hash(contraseña, saltRounds);

const nuevoUsuario = { nombre, contraseña: hash, rol };

return nuevoUsuario; // Simulación de guardado en DB

}

**3.2 Inicio de sesión (JWT con expiración y rol)**

const jwt = require('jsonwebtoken');

async function loginUsuario(nombre, contraseñaIngresada, usuarioRegistrado) {

const ok = await bcrypt.compare(contraseñaIngresada, usuarioRegistrado.contraseña);

if (!ok) throw new Error('Credenciales incorrectas');

return jwt.sign(

{ id: usuarioRegistrado.id, rol: usuarioRegistrado.rol },

process.env.JWT\_SECRET,

{ expiresIn: '1h' }

);

}

**3.3 Middleware de autenticación**

function verificarToken(req, res, next) {

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

if (!token) return res.status(403).send('Acceso denegado');

jwt.verify(token, process.env.JWT\_SECRET, (err, decoded) => {

if (err) return res.status(403).send('Token inválido o expirado');

req.user = decoded;

next();

});

}

**3.4 Control de acceso según rol**

function accesoAdministrador(req, res, next) {

if (req.user.rol !== 'administrador') return res.status(403).send('Acceso restringido');

next();

}

function accesoEditor(req, res, next) {

if (req.user.rol === 'usuario') return res.status(403).send('Acceso restringido');

next();

}

**3.5 Rutas protegidas**

app.get('/proyectos', verificarToken, accesoEditor, (req, res) => res.send('Lista de proyectos'));

app.post('/proyectos', verificarToken, accesoEditor, (req, res) => res.send('Proyecto creado'));

app.delete('/proyectos/:id', verificarToken, accesoAdministrador, (req, res) => res.send('Proyecto eliminado'));

**4. Pruebas de Seguridad**

* **Login correcto:** genera JWT válido con rol incluido.
* **Login incorrecto:** acceso denegado.
* **Acceso con token inválido o expirado:** rechazo con error 403.
* **Acciones restringidas:**
  + Usuario común no puede crear/editar ni eliminar.
  + Editor puede crear/editar, pero no eliminar.
  + Administrador con permisos completos.

**5. Recomendaciones de Ciberseguridad**

1. Usar **JWT con expiración corta** + Refresh Tokens para mayor seguridad.
2. Configurar **cookies HttpOnly + Secure + SameSite** en entornos web.
3. Implementar **MFA (autenticación multifactor)** para cuentas administrativas.
4. Aplicar cabeceras de seguridad (**CSP, HSTS, X-Frame-Options**).
5. Monitorear intentos fallidos y aplicar **rate limiting** en login.
6. Establecer una política de contraseñas robustas (mínimo 12 caracteres, complejidad).

**6. Reflexión**

Este ejercicio evidenció la importancia de separar claramente **autenticación** (validar identidad) y **autorización** (controlar permisos). El uso de **bcrypt** garantiza que las contraseñas no se almacenen en texto claro, mientras que los **JWT** permiten manejar sesiones de manera segura y escalable. El control de roles asegura la aplicación del **principio de menor privilegio**, reduciendo riesgos de abuso de permisos.

**7. Conclusión**

La implementación de autenticación segura con **JWT y bcrypt** junto con un control de acceso basado en roles permitió establecer un entorno protegido para la aplicación de gestión de proyectos. Las pruebas confirmaron que cada usuario accede solo a lo que corresponde a su rol, y que las rutas críticas están adecuadamente protegidas. Este laboratorio demuestra que aplicar buenas prácticas de ciberseguridad desde el desarrollo es clave para garantizar la **confidencialidad, integridad y disponibilidad** de la información.