

Parte práctica de la explotación de vulnerabilidades:

App01: <https://github.com/OWASP/NodeGoat>

```

@MacBook-Pro:~/others/NodeGoat$ npm start
> owasp-nodejs-goat@1.3.0 start /Users/.../others/NodeGoat
> node server.js

undefined
Current Config:
{
  port: 4000,
  db: 'mongodb://...',
  cookieSecret: 'session_cookie_secret_key_here',
  cryptoKey: 'a_secure_key_for_crypto_here',
  cryptoAlgo: 'aes256',
  hostname: 'localhost',
  environmentalScripts: [
    '<script>document.write("<script src=\'http://\' + (location.host + ...
  ],
  zapHostname: '192.168.56.20',
  zapPort: '8080',
  zapApiKey: 'v9dn0balpqas1pcc281tn5ood1',
  zapApiFeedbackSpeed: 5000
}
Connected to the database
Express http server listening on port 4000

```

A1 - 1 Server Side JS Injection

Cuando eval() se utilizan para procesar las entradas proporcionadas por el usuario, un atacante puede explotarlo para inyectar y ejecutar código JavaScript malicioso en el servidor.

En la práctica vamos pasar el comando 'process.exit()' en el campo donde se utiliza la función 'eval()'

El servidor ha ejecutado y parado la aplicación.

=>

Para arreglar la vulnerabilidad lo mejor es evitar la función eval() para analizar los datos de entrada del usuario.

En el atributo 'preTax' cambié la función eval() por el parseInt() y también añadí el "use strict"; al principio de la función.

```
this.handleContributionsUpdate = (req, res, next) => {
  /*jslint evil: true */
  // Insecure use of eval() to parse inputs
  const preTax = parseInt(req.body.preTax);
  const afterTax = eval(req.body.afterTax);
  const roth = eval(req.body.roth);
  // ...
}
```

=>

Invalid contribution percentages

This screen allows you to change the payroll percentages deducted from your paycheck for each contribution type.

Contribution Type	Payroll Contribution Percent (per pay period)	New Payroll Contribution Percent (per pay period)
Employee Pre-Tax	0 %	<input type="text" value="process.exit()"/> %
Roth Contribution	0 %	<input type="text" value="0"/> %
Employee After Tax	0 %	<input type="text" value="0"/> %

A3-Cross-Site Scripting (XSS)

Las fallas XSS ocurren cada vez que una aplicación toma datos que no son de confianza y los envía a un navegador web sin la validación o el escape adecuados

Añadiendo un código sin cualquiera validación en el formulario

<script>alert(decodeURIComponent(document.location))</script>

My Profile

Profile updated successfully.

[Edit Profile](#)

First Name

Last Name

SSN

=>

localhost:5000 says
http://localhost:5000/allocations/2

Para arreglar la falla use the 'Auto scaping' para desactivar el código HTML.

```
// Template system setup
swig.setDefaults({
  // Autoescape disabled
  // Fix for A3 - XSS, enable auto escaping
  root: __dirname + "/app/views",
  autoescape: true, // default value
});
```

=>

Profile updated successfully.

[Edit Profile](#)

First Name

Last Name

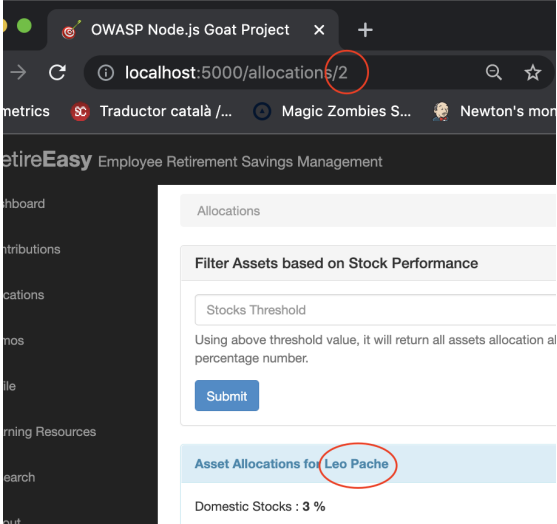
SSN

User evil <script>alert(decodeURIComponent(document.location))</script>

A4-Insecure Direct Object References

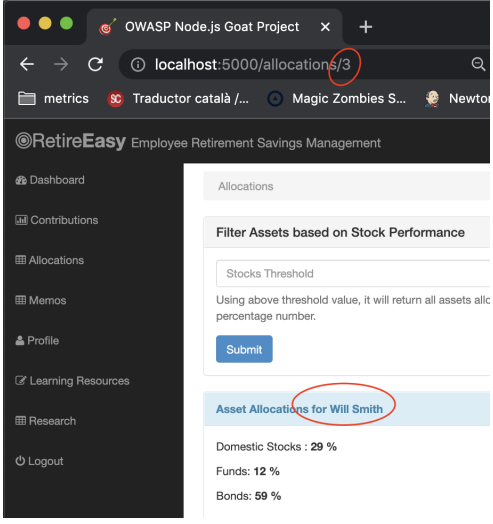
Para ejemplificar un caso de referencia a objetos directos tenemos el servicio que hace la búsqueda del usuario con un GET y envía el ID de la base de datos en la URL que sin tratamiento se puede cambiar y acceder a datos de otros usuarios.

http://localhost:5000/allocations/2



=>

http://localhost:5000/allocations/3



Una posible solución es recuperar el usuario que inició sesión (usando req.session.userId) y no simplemente aceptar lo que viene de la llamada.

Siempre que el servicio de Alocaciones es llamado se limita el acceso a los datos del usuario que está logueado.

```
this.displayAllocations = (req, res, next) => {  
  // Fix for A4 Insecure DOR - take user id from session instead of from URL param  
  const { userId } = req.session;  
  const {  
    threshold  
  } = req.query  
}
```

=>

