

Contents

1 SEC-009: Exposición de Datos Sensibles en Código/Logs	1
1.1 Severidad	1
1.2 Referencia OWASP	1
1.3 Referencias CWE	1
1.4 Descripción	1
1.5 Mapeo de Cumplimiento	1
1.6 Ejemplo de Código Vulnerable	2
1.7 Implementación Segura	4
1.8 Pasos de Remediación	8
1.9 Referencias	9

1 SEC-009: Exposición de Datos Sensibles en Código/Logs

1.1 Severidad

Alta 

1.2 Referencia OWASP

A02:2021 - Fallos Criptográficos

1.3 Referencias CWE

- CWE-798: Uso de Credenciales Codificadas
- CWE-532: Inserción de Información Sensible en Archivo de Log

1.4 Descripción

Las vulnerabilidades de Exposición de Datos Sensibles ocurren cuando una aplicación expone inadvertidamente información confidencial a través de código fuente, archivos de log, mensajes de error, archivos temporales o archivos de configuración. En servidores MCP, esto incluye exposición de claves API, credenciales de base de datos, claves de cifrado privadas, información personal o datos críticos del negocio. Los atacantes pueden extraer esta información a través de repositorios de código, análisis de logs, archivos de copia de seguridad o páginas de error, llevando a acceso no autorizado, brechas de datos o compromiso del sistema.

1.5 Mapeo de Cumplimiento

1.5.1 SOC2

- **CC6.1:** Controles de Acceso Lógico y Físico - La organización restringe el acceso a datos sensibles en código y logs.
- **CC6.2:** Control de Acceso - Las credenciales sensibles se gestionan a través de mecanismos seguros, no codificadas.

- **CC7.2:** Monitoreo del Sistema - La organización monitorea logs y previene exposición de datos sensibles.

1.5.2 HIPAA

- **§164.312(a)(1):** Control de Acceso - Restringir acceso a ePHI en logs y código fuente.
- **§164.312(d):** Cifrado y Descifrado - Cifrar datos sensibles en tránsito y en reposo en logs.
- **§164.312(b):** Controles de Auditoría - Implementar mecanismos para registrar acceso a información sensible.

1.5.3 PCI DSS

- **2.1:** Los datos sensibles no deben almacenarse en código fuente o logs
- **6.5.10:** Los archivos de log y mensajes de error no deben contener datos de tarjeta de pago o credenciales sensibles
- **8.2.3:** Las contraseñas no deben mostrarse ni almacenarse en logs
- **11.5.2:** Implementar alertas para intentos no autorizados de acceder a datos sensibles

1.6 Ejemplo de Código Vulnerable

// ☐ INSEGURO: Credenciales codificadas y datos sensibles en logs

```
const express = require('express');
const mysql = require('mysql2');
const app = express();
```

```
app.use(express.json());
```

// PELIGRO: Credenciales de base de datos codificadas en el código

```
const db = mysql.createConnection({
  host: 'localhost',
  user: 'admin',
  password: 'MyP@ssw0rd123',
  database: 'mcp_production'
});
```

// PELIGRO: Clave API codificada en código fuente

```
const STRIPE_API_KEY = 'sk_live_51H8X5E2eZvKLC2p5nF9mK0L2p3q4r5s6t7u8v9w0x1y2z3a4b';
```

// Endpoint vulnerable: login de usuario con logging sensible

```
app.post('/api/login', (req, res) => {
  const { username, password, mfa_code } = req.body;
```

// PELIGRO: Logging de credenciales en texto plano

```
console.log(`Login attempt: ${username} with password ${password}`);
```

```

// Verificar credenciales
db.query(
  'SELECT * FROM users WHERE username = ? AND password = ?',
  [username, password],
  (err, results) => {
    if (err) {
      // PELIGRO: Exposición de detalles de error de base de datos
      console.error('Database error:', err);
      return res.status(500).json({
        error: err.message,
        details: err.sqlState
      });
    }

    if (results.length > 0) {
      // PELIGRO: Logging de códigos MFA
      console.log(`MFA code for user ${username}: ${mfa_code}`);
      res.json({ success: true, user: results[0] });
    } else {
      res.status(401).json({ error: 'Credenciales inválidas' });
    }
  }
);
});

// Endpoint vulnerable: procesamiento de pago
app.post('/api/payment', (req, res) => {
  const { amount, cardNumber, cvv, expiry } = req.body;

  // PELIGRO: Detalles de tarjeta registrados en consola
  console.log(`Processing payment: ${cardNumber} ${expiry}`);

  // PELIGRO: Clave Stripe codificada usada en solicitud
  // Nunca exponer en frontend o logs
  const response = makeStripeRequest(amount, cardNumber, STRIPE_API_KEY);

  res.json(response);
});

// Endpoint vulnerable: perfil de usuario con PII
app.get('/api/user/:id', (req, res) => {
  const userId = req.params.id;

  db.query(
    'SELECT * FROM users WHERE id = ?',
    [userId],
    (err, results) => {

```

```

    if (err) {
      return res.status(500).json({ error: err.message });
    }

    // PELIGRO: Exposición de todos los datos de usuario incluyendo SSN, teléfono
    res.json(results[0]);
  }
);
});

app.listen(3000);

```

1.7 Implementación Segura

```

// □ SEGURO: Credenciales en variables de entorno, datos sensibles protegidos
const express = require('express');
const mysql = require('mysql2/promise');
const winston = require('winston');
const app = express();

app.use(express.json());

// Cargar credenciales desde variables de entorno (nunca hardcodear)
const db = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD, // Desde .env o gestor de secretos
  database: process.env.DB_NAME,
  waitForConnections: true,
  connectionLimit: 10
});

const STRIPE_API_KEY = process.env.STRIPE_API_KEY; // Del gestor de secretos
const LOG_LEVEL = process.env.LOG_LEVEL || 'info';

// Configurar logger para excluir datos sensibles
const logger = winston.createLogger({
  level: LOG_LEVEL,
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' })
  ]
});

// Middleware para sanitizar cuerpo de solicitud antes de registrar
const sanitizeRequest = (req, res, next) => {

```

```

req.sanitized = JSON.parse(JSON.stringify(req.body));

// Eliminar campos sensibles de copia sanitizada
const sensitiveFields = ['password', 'mfa_code', 'cardNumber', 'cvv', 'expiry',
sensitiveFields.forEach(field => {
  if (req.sanitized[field]) {
    req.sanitized[field] = '[REDACTED]';
  }
});

next();
};

app.use(sanitizeRequest);

// Endpoint seguro: login de usuario con logging seguro
app.post('/api/login', async (req, res) => {
  const { username, password, mfa_code } = req.body;

  if (!username || !password) {
    return res.status(400).json({ error: 'Credenciales faltantes' });
  }

  try {
    // Registrar solo información no sensible
    logger.info(`Login attempt`, {
      username: username,
      timestamp: new Date().toISOString(),
      ip: req.ip
    });

    // Consultar base de datos
    const [rows] = await db.execute(
      'SELECT id, username, password_hash, mfa_enabled FROM users WHERE username =
      [username]
    );

    if (rows.length === 0) {
      logger.warn(`Failed login: user not found`, {
        username: username,
        ip: req.ip
      });
      return res.status(401).json({ error: 'Credenciales inválidas' });
    }

    const user = rows[0];

    // Verificar contraseña (no registrarla)

```

```

const bcrypt = require('bcrypt');
const isValidPassword = await bcrypt.compare(password, user.password_hash);

if (!isValidPassword) {
  logger.warn(`Failed login: invalid password`, {
    username: username,
    ip: req.ip
  });
  return res.status(401).json({ error: 'Credenciales inválidas' });
}

// Verificar MFA si está habilitado (no registrar código)
if (user.mfa_enabled && !verifyMFA(user.id, mfa_code)) {
  logger.warn(`Failed login: invalid MFA`, {
    username: username,
    ip: req.ip
  });
  return res.status(401).json({ error: 'Código MFA inválido' });
}

logger.info(`Successful login`, {
  username: username,
  userId: user.id
});

// No devolver hash de contraseña
delete user.password_hash;
res.json({ success: true, user });
} catch (error) {
  // Registrar error sin exponer detalles
  logger.error(`Login error`, {
    error: error.name,
    timestamp: new Date().toISOString()
  });
  res.status(500).json({ error: 'Autenticación fallida' });
}
});

// Endpoint seguro: procesamiento de pago sin registrar datos sensibles
app.post('/api/payment', async (req, res) => {
  const { amount, token } = req.body; // Usar token de tarjeta, no datos brutos

  if (!amount || !token) {
    return res.status(400).json({ error: 'Falta información de pago' });
  }

  try {

```

```

// Nunca manejar datos de tarjeta brutos en la aplicación
// Usar tokens del procesador de pago o formularios alojados

logger.info(`Payment processing initiated`, {
  amount: amount,
  timestamp: new Date().toISOString(),
  ip: req.ip
  // Nota: token no se registra
});

// Hacer solicitud de pago (token enviado a Stripe, no datos de tarjeta brutos)
const stripeResponse = await makeStripeRequest(amount, token, STRIPE_API_KEY);

if (stripeResponse.success) {
  logger.info(`Payment successful`, {
    transactionId: stripeResponse.transaction_id,
    amount: amount
  });

  res.json({ success: true, transactionId: stripeResponse.transaction_id });
} else {
  logger.warn(`Payment failed`, {
    reason: stripeResponse.reason,
    amount: amount
  });
  res.status(400).json({ error: 'Pago fallido' });
}

} catch (error) {
  logger.error(`Payment processing error`, {
    error: error.name,
    timestamp: new Date().toISOString()
  });
  res.status(500).json({ error: 'Procesamiento de pago fallido' });
}
});

// Endpoint seguro: perfil de usuario con protección de PII
app.get('/api/user/:id', async (req, res) => {
  const userId = req.params.id;

  // Validar que el usuario solicita sus propios datos
  if (req.user.id !== parseInt(userId)) {
    logger.warn(`Unauthorized profile access attempt`, {
      requestingUser: req.user.id,
      targetUser: userId,
      ip: req.ip
    });
  }
}

```

```

    return res.status(403).json({ error: 'Forbidden' });
  }

  try {
    const [rows] = await db.execute(
      'SELECT id, username, email, created_at FROM users WHERE id = ?',
      [userId]
    );

    if (rows.length === 0) {
      return res.status(404).json({ error: 'Usuario no encontrado' });
    }

    // Nunca exponer SSN, teléfono u otros campos sensibles en respuesta API
    logger.info(`Profile retrieved`, {
      userId: userId
    });

    res.json(rows[0]);
  } catch (error) {
    logger.error(`Profile retrieval error`, {
      error: error.name
    });
    res.status(500).json({ error: 'Fallo al recuperar perfil' });
  }
});

app.listen(3000);

```

1.8 Pasos de Remediación

1. **Almacenar Credenciales Sensibles en Variables de Entorno o Gestor de Secretos:** Nunca codificar claves API, credenciales de base de datos o claves de cifrado en código fuente. Usar variables de entorno (archivos .env en desarrollo, gestores de secretos en producción como AWS Secrets Manager, HashiCorp Vault o Azure Key Vault). Rotar credenciales regularmente. Implementar detección automática de secretos en repositorios de código usando herramientas como GitGuardian o TruffleHog.
2. **Implementar Sanitización Comprehensive de Logs:** Configurar frameworks de logging para excluir campos sensibles (contraseñas, tokens, SSN, tarjetas de crédito, códigos MFA). Implementar middleware para sanitizar cuerpos de solicitud/respuesta antes de registrar. Usar logging estructurado con filtrado a nivel de campo. Nunca registrar mensajes de error completos que podrían revelar información del sistema - registrar solo tipos de error e IDs de error seguros.
3. **Aplicar Clasificación de Datos y Minimización:** Clasificar datos como sensi-

bles (PII, credenciales, datos de pago) y no sensibles. Solo recopilar y registrar datos mínimos necesarios. No exponer campos sensibles en respuestas API - devolver solo lo que el cliente necesita. Implementar controles de acceso a nivel de campo en respuestas basadas en roles de usuario.

4. **Implementar Mecanismos de Detección y Respuesta:** Usar sistemas SIEM (Security Information and Event Management) para detectar patrones de acceso sospechosos. Implementar alertas para intentos de login fallidos repetidos, acceso de datos inusual o intentos de acceder a endpoints sensibles. Realizar revisiones de código regulares enfocadas en secretos codificados. Implementar escaneo automatizado de repositorios para credenciales expuestas. Monitorear políticas de retención de logs para asegurar que los logs antiguos que contienen datos sensibles se eliminen apropiadamente.

1.9 Referencias

- [OWASP Sensitive Data Exposure](#)
- [CWE-798: Uso de Credenciales Codificadas](#)
- [CWE-532: Inserción de Información Sensible en Archivo de Log](#)
- [OWASP Logging Cheat Sheet](#)
- [Node.js Credential Security](#)

Última Actualización: Enero 2026

Estado: Publicado

Idioma: Español