Contents

1 SEC-008: Fuga de Datos en Respuestas

1.1 Severidad

Media

1.2 Referencia OWASP

A01:2021 - Control de Acceso Roto

1.3 Referencias CWE

- CWE-200: Exposición de Información Sensible a un Actor No Autorizado
- CWE-209: Exposición de Información a través de un Mensaje de Error

1.4 Descripción

La Fuga de Datos en Respuestas ocurre cuando una API devuelve más información sensible de la necesaria, exponiendo datos no autorizados a los clientes. Esto incluye devolver registros de usuario completos cuando solo se necesita el email, exponer detalles internos del sistema en respuestas, devolver timestamps que revelan comportamiento del sistema o incluir stack traces en respuestas de error. En servidores MCP, esto puede exponer lógica empresarial, identificadores internos o información sensible del usuario a partes no autorizadas.

1.5 Mapeo de Cumplimiento

1.5.1 SOC2

- **CC6.1**: Controles de Acceso Lógico y Físico - Los datos se clasifican y solo se expone información necesaria.
- **CC6.2**: Control de Acceso - Los controles de acceso a nivel de campo restringen la exposición de datos sensibles.

- **CC7.2**: Monitoreo del Sistema - Las respuestas se monitorean para detectar fugas de datos no intencionadas.

1.5.2 HIPAA

- **§164.312(a)(1)**: Control de Acceso - Devolver solo la ePHI mínima necesaria en respuestas API.
- **§164.312(c)(1)**: Controles de Integridad - Verificar que solo datos autorizados se devuelven a usuarios.

1.5.3 PCI DSS

- **6.5.5**: Las referencias de objeto directo inseguras se previenen validando exposición de datos
- **3.2.1**: Los datos del titular de la tarjeta no se almacenan ni devuelven en respuestas API
- **10.1**: Implementar logging de todo acceso para auditar qué datos se devuelven

1.6 Ejemplo de Código Vulnerable

```
// □ INSEGURO: Devolver datos sensibles innecesarios
const express = require('express');
const app = express();

// Endpoint vulnerable: devolver todos los campos de usuario
app.get('/api/users/:id', (req, res) => {
  const user = {
    id: 123,
    username: 'john_doe',
    email: 'john@example.com',
    password_hash: '$2b$12$...', // ¡Expuesto!
    social_security_number: '123-45-6789', // ¡Expuesto!
    credit_card: '4532-1234-5678-9012', // ¡Expuesto!
    internal_employee_id: 'EMP-12345', // ¡Expuesto!
    api_key: 'sk_live_abc123xyz', // ¡Expuesto!
    home_address: '123 Main St', // ¡Expuesto!
    phone: '555-1234' // ¡Expuesto!
  };

  res.json(user); // ¡Devuelve todo!
});

// Endpoint vulnerable: error con stack trace
app.get('/api/data', (req, res) => {
  try {
    throw new Error('Database connection failed');
  } catch (error) {
```

```
    // PELIGRO: Stack trace revela rutas internas
    res.status(500).json({
      error: error.message,
      stack: error.stack // Expone rutas de archivo y nombres de función
    });
  }
});

// Endpoint vulnerable: listar todos los usuarios con datos sensibles
app.get('/api/users', (req, res) => {
  const users = [
    { id: 1, username: 'admin', password_hash: '...', role: 'admin', ssn: '111-11-
    { id: 2, username: 'user', password_hash: '...', role: 'user', ssn: '222-22-22
  ];

  res.json(users); // ¡Todos los campos sensibles expuestos!
});

app.listen(3000);
```

1.7 Implementación Segura

```
// □ SEGURO: Devolver solo datos necesarios y no sensibles
const express = require('express');
const app = express();

// Definir esquemas de respuesta para diferentes contextos
const userSchemas = {
  // Perfil público (lo que otros usuarios pueden ver)
  publicProfile: (user) => ({
    id: user.id,
    username: user.username,
    created_at: user.created_at
  }),

  // Perfil privado (lo que el usuario autenticado puede ver sobre sí mismo)
  privateProfile: (user) => ({
    id: user.id,
    username: user.username,
    email: user.email,
    created_at: user.created_at,
    updated_at: user.updated_at
  }),

  // Vista de administrador (lo que los admins pueden ver)
  adminView: (user) => ({
    id: user.id,
```

```
    username: user.username,
    email: user.email,
    role: user.role,
    created_at: user.created_at,
    last_login: user.last_login,
    status: user.status
  }),

  // Vista de lista (datos mínimos para listados)
  listView: (user) => ({
    id: user.id,
    username: user.username,
    status: user.status
  })
};

// Endpoint seguro: devolver datos de usuario filtrados
app.get('/api/users/:id', (req, res) => {
  const userId = parseInt(req.params.id);

  if (isNaN(userId)) {
    return res.status(400).json({ error: 'ID de usuario inválido' });
  }

  // Simular obtención de usuario
  const user = {
    id: userId,
    username: 'john_doe',
    email: 'john@example.com',
    password_hash: '$2b$12$...',
    ssn: '123-45-6789',
    role: 'user',
    created_at: '2024-01-01T00:00:00Z'
  };

  // Determinar qué esquema usar basado en contexto de autenticación
  let responseData;

  if (req.user.id === userId) {
    // Usuario viendo su propio perfil
    responseData = userSchemas.privateProfile(user);
  } else if (req.user.role === 'admin') {
    // Admin viendo otro usuario
    responseData = userSchemas.adminView(user);
  } else {
    // Otros usuarios viendo perfil público
    responseData = userSchemas.publicProfile(user);
  }
```

```
  res.json(responseData);
});

// Endpoint seguro: listar usuarios con datos mínimos
app.get('/api/users', (req, res) => {
  const users = [
    { id: 1, username: 'admin', role: 'admin', created_at: '2024-01-01' },
    { id: 2, username: 'user', role: 'user', created_at: '2024-01-02' }
  ];

  // Devolver solo datos de vista de lista
  const filtered = users.map(u => userSchemas.listView(u));

  res.json({
    total: filtered.length,
    items: filtered
  });
});

// Endpoint seguro: manejo de errores sin exponer detalles
app.get('/api/data', (req, res) => {
  try {
    throw new Error('Database connection failed');
  } catch (error) {
    // Registrar error completo del lado del servidor
    console.error('Full error details:', error);

    // Devolver error genérico al cliente
    res.status(500).json({
      error: 'Error interno del servidor',
      errorId: 'ERR_DB_001' // Puede usarse para tickets de soporte
    });
  }
});

// Endpoint seguro: búsqueda sin exponer todos los datos
app.get('/api/search', (req, res) => {
  const { query } = req.query;

  if (!query) {
    return res.status(400).json({ error: 'Consulta de búsqueda requerida' });
  }

  // Simular resultados de búsqueda
  const results = [
    { id: 1, username: 'john_doe', created_at: '2024-01-01' },
    { id: 2, username: 'jane_doe', created_at: '2024-01-02' }
```

```
  ];

  // Devolver solo campos seguros para resultados de búsqueda
  const filtered = results.map(u => ({
    id: u.id,
    username: u.username,
    // No incluir timestamps que podrían revelar información
  }));

  res.json({
    query: query,
    results: filtered,
    count: filtered.length
  });
});

app.listen(3000);
```

1.8 Pasos de Remediación

1. **Definir Esquemas de Respuesta por Contexto**: Crear esquemas de respuesta explícitos para diferentes casos de uso (vista pública, vista privada, vista de admin, vista de lista). Solo incluir campos necesarios para cada contexto. Usar controles de acceso a nivel de campo para determinar qué datos exponer. Documentar qué datos se exponen en cada respuesta de API.
2. **Implementar Filtrado de Campos en Recuperación de Datos**: Filtrar campos sensibles a nivel de consulta de base de datos cuando sea posible. Usar proyecciones ORM para seleccionar solo columnas necesarias. Nunca obtener datos innecesarios y luego filtrarlos en código. Auditar consultas de base de datos para asegurar recuperación mínima de datos.
3. **Manejar Errores de Forma Segura**: Nunca exponer stack traces, rutas de archivo o detalles internos de error a los clientes. Devolver mensajes de error genéricos a los clientes mientras se registran detalles completos del lado del servidor. Crear IDs de error para rastreo/soporte sin revelar detalles del sistema. Sanitizar todos los mensajes de error antes de devolver a los clientes.
4. **Monitorear y Registrar Exposición de Datos**: Implementar logging de todas las respuestas API (datos sensibles redactados). Usar monitoreo para detectar tamaños de respuesta inusuales o campos de datos inesperados. Auditar regularmente respuestas API para fuga de datos no intencionada. Implementar limitación de tasa en endpoints de lectura para prevenir cosecha de datos.

1.9 Referencias

- OWASP API Security - Excessive Data Exposure
- CWE-200: Exposición de Información Sensible
- OWASP API Response Security

Última Actualización: Enero 2026
Estado: Publicado
Idioma: Español