Contents

1 SEC-005: Entidad Externa XML (XXE)

1.1 Severidad

**Alta **

1.2 Referencia OWASP

A05:2021 - Configuración de Seguridad Incorrecta

1.3 Referencias CWE

- CWE-611: Restricción Impropia de Referencia de Entidad Externa XML

1.4 Descripción

Las vulnerabilidades de inyección de Entidad Externa XML (XXE) ocurren cuando una aplicación analiza entrada XML no confiable sin deshabilitar el procesamiento de entidades externas. Esto permite a los atacantes leer archivos arbitrarios del servidor, realizar ataques de Falsificación de Solicitud del Lado del Servidor (SSRF), conducir ataques de denegación de servicio o lograr ejecución remota de código. En servidores MCP que aceptan cargas/importaciones de XML, SOAP o SVG, las vulnerabilidades XXE pueden llevar al compromiso completo del sistema y exfiltración de datos.

1.5 Mapeo de Cumplimiento

1.5.1 SOC2

- **CC6.1**: Controles de Acceso Lógico y Físico - La organización implementa controles de acceso para sistemas de archivos y datos sensibles.
- **CC6.6**: Operaciones del Sistema - La organización previene la ejecución no autorizada de código potencialmente malicioso.
- **CC7.2**: Monitoreo del Sistema - La organización detecta y monitorea intentos de parsing de XML y eventos de seguridad.

1.5.2 HIPAA

- **§164.312(a)(1)**: Control de Acceso - Implementar políticas técnicas para prevenir acceso no autorizado a ePHI en archivos.
- **§164.312(c)(1)**: Controles de Integridad - Proteger ePHI de alteración no autorizada mediante procesamiento seguro de XML.
- **§164.312(e)(1)**: Seguridad de Transmisión - Implementar controles para prevenir divulgación de ePHI a través de explotación XXE.

1.5.3 PCI DSS

- **6.2.4**: Todo el software personalizado debe desarrollarse de forma segura con protecciones de procesamiento XML
- **6.5.1**: Las fallas de inyección incluyendo XXE se abordan mediante prácticas de codificación segura
- **6.5.4**: Las referencias de objeto directo inseguras vía XXE se previenen
- **11.3.1**: Las vulnerabilidades XXE se identifican mediante escaneo regular de vulnerabilidades

1.6 Ejemplo de Código Vulnerable

```
// □ INSEGURO: Parsing de XML sin deshabilitar entidades externas
const express = require('express');
const xml2js = require('xml2js');
const libxmljs = require('libxmljs');
const app = express();

app.use(express.text({ type: 'application/xml' }));

// Endpoint vulnerable: importación de datos XML usando xml2js
app.post('/api/import-config', async (req, res) => {
  const xmlData = req.body;

  // PELIGRO: El parser predeterminado de xml2js permite ataques XXE
  // Entrada del atacante: <?xml version="1.0"?>
  // <!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
  // <config><user>&xxe;</user></config>
  const parser = new xml2js.Parser();

  try {
    const result = await parser.parseStringPromise(xmlData);
    res.json({ success: true, config: result });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// Endpoint vulnerable: procesamiento de carga SVG
```

```
app.post('/api/upload-svg', async (req, res) => {
  const svgData = req.body;

  // PELIGRO: libxmljs por defecto permite procesamiento de entidades externas
  try {
    const xmlDoc = libxmljs.parseXml(svgData);
    const root = xmlDoc.root();
    res.json({ success: true, svg: root.toString() });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// Endpoint vulnerable: manejador de solicitud SOAP
app.post('/api/soap-service', async (req, res) => {
  const soapRequest = req.body;

  // PELIGRO: Sin protección XXE para mensajes SOAP
  // El atacante puede incrustar payloads XXE en solicitudes SOAP
  try {
    const parser = new xml2js.Parser();
    const parsed = await parser.parseStringPromise(soapRequest);
    res.json({ response: 'SOAP procesado' });
  } catch (error) {
    res.status(500).json({ error: 'Procesamiento fallido' });
  }
});

app.listen(3000);
```

1.7 Implementación Segura

```
// □ SEGURO: Parsing de XML con protección XXE habilitada
const express = require('express');
const xml2js = require('xml2js');
const libxmljs = require('libxmljs');
const app = express();

app.use(express.text({ type: 'application/xml', limit: '1mb' }));

// Endpoint seguro: importación de datos XML con XXE deshabilitado
app.post('/api/import-config', async (req, res) => {
  const xmlData = req.body;

  // Validar tamaño de XML para prevenir ataque de mil risas
  if (xmlData.length > 1024 * 1024) { // máximo 1MB
    return res.status(400).json({ error: 'XML demasiado grande' });
```

```
  }

  try {
    // Crear parser con protecciones XXE
    const parser = new xml2js.Parser({
      // Deshabilitar procesamiento de entidades externas - configuración más impo
      processEntities: false,

      // Deshabilitar procesamiento DOCTYPE para prevenir XXE
      doctype: false,

      // Prevenir ataque de mil risas DoS
      maxDepth: 20,

      // Limitar expansión de entidades
      maxAttributes: 50,

      // Deshabilitar procesamiento de namespaces que podría ser explotado
      xmlns: false,

      // Recortar valores para prevenir explotación de espacios en blanco
      trim: true
    });

    const result = await parser.parseStringPromise(xmlData);

    // Validación adicional de estructura parseada
    if (!result.config) {
      return res.status(400).json({ error: 'Estructura XML inválida' });
    }

    res.json({ success: true, config: result });

  } catch (error) {
    console.error('Error de parsing XML:', error);
    res.status(400).json({ error: 'Formato XML inválido' });
  }
});

// Endpoint seguro: carga de SVG con protección XXE
app.post('/api/upload-svg', async (req, res) => {
  const svgData = req.body;

  // Validar tamaño
  if (svgData.length > 5 * 1024 * 1024) { // máximo 5MB para SVG
    return res.status(400).json({ error: 'SVG demasiado grande' });
  }
```

```
// Validar que comienza con etiqueta SVG
if (!svgData.includes('<svg')) {
  return res.status(400).json({ error: 'No es un archivo SVG válido' });
}

try {
  // Crear parser con protecciones XXE deshabilitadas
  const parser = new libxmljs.SaxParser((cb) => {
    cb.onStartElement((elem, attrs) => {
      // Aceptar inicio de elemento
    });
    cb.onError((msg) => {
      throw new Error(`Error de parse XML: ${msg}`);
    });
  });

  // Habilitar protección XXE en libxmljs deshabilitando procesamiento DTD
  const options = {
    dtdload: false,           // No cargar DTDs externos
    dtdvalid: false,          // No validar contra DTDs
    noent: false,             // No expandir entidades
    nonet: true,              // No acceder a red
    nocdata: false            // OK procesar secciones CDATA
  };

  // Alternativa: Usar parsing XML más seguro con validación
  const xmlDoc = libxmljs.parseXml(svgData, {
    dtdload: false,
    dtdvalid: false,
    noent: false,
    nonet: true
  });

  // Validar que el elemento raíz es svg
  const root = xmlDoc.root();
  if (root.name() !== 'svg') {
    return res.status(400).json({ error: 'El elemento raíz debe ser <svg>' });
  }

  // Eliminar elementos potencialmente peligrosos
  const dangerousElements = ['script', 'iframe', 'embed', 'object'];
  dangerousElements.forEach(elem => {
    const elements = xmlDoc.find(`//${elem}`);
    elements.forEach(e => e.remove());
  });

  res.json({
    success: true,
```

```
      svg: root.toString().substring(0, 10000) // Limitar tamaño de respuesta
    });

  } catch (error) {
    console.error('Error de parsing SVG:', error);
    res.status(400).json({ error: 'Formato SVG inválido' });
  }
});

// Endpoint seguro: manejador de solicitud SOAP con protección XXE
app.post('/api/soap-service', async (req, res) => {
  const soapRequest = req.body;

  // Validar tamaño
  if (soapRequest.length > 2 * 1024 * 1024) { // máximo 2MB
    return res.status(400).json({ error: 'Mensaje SOAP demasiado grande' });
  }

  // Verificar patrones XXE en XML en bruto (defensa en profundidad)
  const xxePatterns = [
    /<!ENTITY\s+/i,
    /SYSTEM\s+/i,
    /PUBLIC\s+/i,
    /<!DOCTYPE\s+/i,
    /xsi:schemaLocation/i
  ];

  for (const pattern of xxePatterns) {
    if (pattern.test(soapRequest)) {
      return res.status(400).json({ error: 'XML sospechoso detectado' });
    }
  }

  try {
    const parser = new xml2js.Parser({
      processEntities: false,
      doctype: false,
      maxDepth: 20,
      maxAttributes: 50,
      xmlns: false,
      trim: true,
      strict: true // Modo estricto para rechazar XML malformado
    });

    const parsed = await parser.parseStringPromise(soapRequest);

    // Validar estructura de envelope SOAP
    if (!parsed['soap:Envelope'] && !parsed.Envelope) {
```

```
      return res.status(400).json({ error: 'Mensaje SOAP inválido' });
    }

    res.json({
      success: true,
      response: 'SOAP procesado de forma segura',
      envelope: Object.keys(parsed)[0]
    });

  } catch (error) {
    console.error('Error de procesamiento SOAP:', error);
    res.status(400).json({ error: 'Procesamiento SOAP fallido' });
  }
});

// Endpoint seguro: endpoint de validación XML genérico
app.post('/api/validate-xml', async (req, res) => {
  const xmlData = req.body;

  // Validación de tamaño
  if (xmlData.length > 5 * 1024 * 1024) { // máximo 5MB
    return res.status(400).json({ error: 'XML demasiado grande' });
  }

  // Detección de XXE basada en patrones
  const xxeIndicators = [
    /<!ENTITY.*SYSTEM/is,
    /<!ENTITY.*PUBLIC/is,
    /<!DOCTYPE/i,
    /SYSTEM\s+["'](file|http|ftp):\/\//i
  ];

  for (const indicator of xxeIndicators) {
    if (indicator.test(xmlData)) {
      return res.status(400).json({
        error: 'Vulnerabilidad XXE detectada',
        message: 'XML contiene declaraciones de entidades o DOCTYPE - no permitido
      });
    }
  }

  try {
    const parser = new xml2js.Parser({
      processEntities: false,
      doctype: false,
      maxDepth: 20,
      trim: true,
      strict: true
```

```
    });

    await parser.parseStringPromise(xmlData);
    res.json({ valid: true, message: 'XML es válido y seguro' });

  } catch (error) {
    res.status(400).json({
      valid: false,
      error: error.message
    });
  }
});

app.listen(3000);
```

1.8 Pasos de Remediación

1. **Deshabilitar Procesamiento de Entidades Externas en Todos los Parsers XML**: Configurar todas las librerías de parsing XML para deshabilitar procesamiento de DTD, expansión de entidades externas y parsing de DOCTYPE. Para xml2js, establecer `processEntities: false` y `doctype: false`. Para libxmljs, establecer `dtdload: false`, `dtdvalid: false` y `noent: false`. Para parsers incorporados de Node.js, usar opciones seguras o actualizar a versiones con valores predeterminados seguros.
2. **Implementar Detección y Prevención de XXE a Nivel de Entrada**: Escanear entrada XML en busca de patrones sospechosos (declaraciones DOCTYPE, declaraciones ENTITY, palabras clave SYSTEM/PUBLIC) antes de analizar. Rechazar XML que contenga estos indicadores. Usar modo de parsing XML estricto que rechace entrada malformada. Implementar restricciones de carga de archivos (validación de tipo de archivo, límites de tamaño, validación de formato).
3. **Usar Validación de Lista Blanca y Procesamiento XML Seguro**: Definir esquemas XML esperados usando librerías de validación XSD. Validar contra esquemas antes de procesar. Usar solo características XML necesarias - deshabilitar características como DTDs externos, namespaces o CDATA si no son necesarios. Considerar usar alternativas más seguras a XML (JSON) cuando sea apropiado para el caso de uso.
4. **Implementar Defensa en Profundidad y Monitoreo**: Ejecutar procesamiento XML en contenedores aislados o entornos sandboxed. Implementar limitación de tasa en endpoints de parsing XML. Monitorear intentos de explotación XXE en logs (acceso a archivos, conexiones de red, mensajes de error). Usar Firewalls de Aplicación Web con reglas de detección XXE. Realizar pruebas de seguridad y revisiones de código regulares de código que maneja XML.

1.9 Referencias

- OWASP XXE Prevention Cheat Sheet

- CWE-611: Restricción Impropia de Referencia de Entidad Externa XML
- PortSwigger: XML External Entity Injection
- OWASP XML Security Cheat Sheet
- Node.js XML Parsing Security Guide

Última Actualización: Enero 2026
Estado: Publicado
Idioma: Español