Contents

1 SEC-004: Falsificación de Solicitud del Lado del Servidor (SSRF)

1.1 Severidad

Alta

1.2 Referencia OWASP

A10:2021 - Falsificación de Solicitud del Lado del Servidor (SSRF)

1.3 Referencias CWE

- CWE-918: Falsificación de Solicitud del Lado del Servidor (SSRF)

1.4 Descripción

Las vulnerabilidades de Falsificación de Solicitud del Lado del Servidor (SSRF) ocurren cuando una aplicación realiza solicitudes HTTP a URLs suministradas por usuarios sin la validación y sanitización adecuadas. En servidores MCP, esto permite a los atacantes hacer que el servidor realice solicitudes no deseadas a servicios internos, endpoints de metadatos de la nube o sistemas externos, potencialmente llevando a acceso no autorizado, divulgación de datos, compromiso de infraestructura en la nube o movimiento lateral dentro de redes internas. Esto es particularmente peligroso en entornos en la nube donde los servicios de metadatos son accesibles a través de HTTP.

1.5 Mapeo de Cumplimiento

1.5.1 SOC2

- **CC6.1**: Controles de Acceso Lógico y Físico - La organización restringe el acceso a recursos y servicios internos.
- **CC6.8**: Operaciones del Sistema - La organización previene acceso no autorizado al sistema a través de comunicaciones de red seguras.

- **CC7.2**: Monitoreo del Sistema - La organización monitorea el tráfico de red y detecta patrones de solicitud sospechosos.

1.5.2 HIPAA

- **§164.312(a)(1)**: Control de Acceso - Implementar políticas técnicas para prevenir acceso no autorizado a sistemas internos que contienen ePHI.
- **§164.312(e)(1)**: Seguridad de Transmisión - Implementar medidas de seguridad técnica para proteger la ePHI de intercepción durante la transmisión.
- **§164.312(c)(1)**: Controles de Integridad - Proteger sistemas internos de ePHI del acceso no autorizado a través de la explotación de SSRF.

1.5.3 PCI DSS

- **6.5.10**: La autenticación rota vía SSRF se previene mediante codificación segura
- **6.2.4**: El software personalizado se desarrolla de forma segura con la validación adecuada de solicitudes
- **1.3.1**: Prohibir acceso directo público a recursos de red interna
- **11.3.1**: Las vulnerabilidades SSRF se identifican mediante escaneo de vulnerabilidades

1.6 Ejemplo de Código Vulnerable

```
// □ INSEGURO: Hacer solicitudes a URLs suministradas por usuarios sin validación
const express = require('express');
const axios = require('axios');
const app = express();

app.use(express.json());

// Endpoint vulnerable: proxy de imagen
app.post('/api/proxy-image', async (req, res) => {
  const { imageUrl } = req.body;

  // PELIGRO: Sin validación de la URL
  // Entrada del atacante: "http://169.254.169.254/latest/meta-data/iam/security-c
  // Esto filtrarían credenciales de AWS del servicio de metadatos
  try {
    const response = await axios.get(imageUrl, { timeout: 5000 });
    res.contentType(response.headers['content-type']);
    res.send(response.data);
  } catch (error) {
    res.status(500).json({ error: 'Fallo al obtener imagen' });
  }
});

// Endpoint vulnerable: vista previa/obtención de URL
```

```javascript
app.post('/api/fetch-content', async (req, res) => {
  const { url } = req.body;

  // PELIGRO: Solicitud directa a URL suministrada por usuario
  // El atacante puede sondear servicios internos: http://localhost:8080/admin
  // O acceder a metadatos de la nube: http://169.254.169.254/
  try {
    const response = await axios.get(url, { timeout: 5000 });
    res.json({ content: response.data.substring(0, 1000) });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Endpoint vulnerable: entrega de webhook
app.post('/api/send-webhook', async (req, res) => {
  const { webhookUrl, payload } = req.body;

  // PELIGRO: Sin validación de la URL del webhook
  // El atacante puede hacer solicitudes a servicios internos con payloads arbitra
  try {
    await axios.post(webhookUrl, payload, { timeout: 5000 });
    res.json({ success: true });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

app.listen(3000);
```

1.7 Implementación Segura

```javascript
// □ SEGURO: Validación estricta de URL y protección de recursos internos
const express = require('express');
const axios = require('axios');
const { URL } = require('url');
const dns = require('dns').promises;
const ipaddr = require('ipaddr.js');
const app = express();

app.use(express.json());

// Rangos de IP bloqueados (direcciones internas y de uso especial)
const BLOCKED_IP_RANGES = [
  '127.0.0.0/8',      // localhost
  '10.0.0.0/8',       // privada
  '172.16.0.0/12',    // privada
```

```
  '192.168.0.0/16',   // privada
  '169.254.0.0/16',   // link-local (¡servicio de metadatos de AWS!)
  '224.0.0.0/4',      // multicast
  '255.255.255.255/32' // broadcast
];

// Dominios en lista blanca para casos de uso específicos
const WHITELISTED_DOMAINS = [
  'cdn.example.com',
  'api.trusted-partner.com',
  'media.example.com'
];

// Función auxiliar para verificar si una IP está en rango bloqueado
function isBlockedIp(ip) {
  try {
    const addr = ipaddr.process(ip);
    return BLOCKED_IP_RANGES.some(range => {
      const [rangeIp, prefixLength] = range.split('/');
      return addr.match(ipaddr.process(rangeIp), parseInt(prefixLength));
    });
  } catch (error) {
    return false;
  }
}

// Función auxiliar para validar URL
async function validateUrl(urlString, whitelist = false) {
  try {
    const parsedUrl = new URL(urlString);

    // Solo permitir HTTP y HTTPS
    if (!['http:', 'https:'].includes(parsedUrl.protocol)) {
      throw new Error('Solo se permiten protocolos HTTP(S)');
    }

    // Si está en modo lista blanca, verificar contra la lista blanca
    if (whitelist && !WHITELISTED_DOMAINS.includes(parsedUrl.hostname)) {
      throw new Error('Dominio no está en la lista blanca');
    }

    // Verificar patrones sospechosos
    if (parsedUrl.hostname.includes('..')) {
      throw new Error('Formato de hostname inválido');
    }

    // Resolución DNS con timeout
    const addresses = await Promise.race([
```

```
      dns.resolve4(parsedUrl.hostname),
      new Promise((_, reject) =>
        setTimeout(() => reject(new Error('Timeout de DNS')), 5000)
      )
    ]);

    // Verificar que las IPs resueltas no estén bloqueadas
    for (const ip of addresses) {
      if (isBlockedIp(ip)) {
        throw new Error(`IP resuelta ${ip} está en rango bloqueado`);
      }
    }

    // Verificar longitud de URL para prevenir URLs extremadamente largas
    if (urlString.length > 2048) {
      throw new Error('URL demasiado larga');
    }

    return parsedUrl;

  } catch (error) {
    throw new Error(`URL inválida: ${error.message}`);
  }
}

// Endpoint seguro: proxy de imagen con validación estricta
app.post('/api/proxy-image', async (req, res) => {
  const { imageUrl } = req.body;

  if (!imageUrl || typeof imageUrl !== 'string') {
    return res.status(400).json({ error: 'imageUrl inválido' });
  }

  try {
    // Validar URL con lista blanca para CDNs confiables
    const parsedUrl = await validateUrl(imageUrl, true);

    // Hacer solicitud con timeout estricto y límites de tamaño
    const response = await axios.get(parsedUrl.toString(), {
      timeout: 5000,
      maxContentLength: 10 * 1024 * 1024, // máximo 10MB
      maxRedirects: 1, // Limitar redirecciones
      validateStatus: (status) => status >= 200 && status < 300 // Solo 2xx
    });

    // Validar tipo de contenido
    const contentType = response.headers['content-type'];
    if (!contentType || !contentType.includes('image/')) {
```

```
      return res.status(400).json({ error: 'Tipo de contenido inválido' });
    }

    res.contentType(contentType);
    res.send(response.data);

  } catch (error) {
    console.error('Error de proxy:', error);
    res.status(400).json({ error: 'No se puede obtener la imagen' });
  }
});

// Endpoint seguro: obtener contenido con acceso restringido
app.post('/api/fetch-content', async (req, res) => {
  const { url } = req.body;

  if (!url || typeof url !== 'string') {
    return res.status(400).json({ error: 'URL inválida' });
  }

  try {
    // Validar URL (sin lista blanca - más permisivo pero aún protegido)
    const parsedUrl = await validateUrl(url, false);

    // Validación adicional: restricciones de puerto
    const port = parsedUrl.port ? parseInt(parsedUrl.port) : (parsedUrl.protocol ==

    // Bloquear puertos de servicios internos comunes
    const BLOCKED_PORTS = [
      22,    // SSH
      3306,  // MySQL
      5432,  // PostgreSQL
      6379,  // Redis
      27017, // MongoDB
      9200,  // Elasticsearch
      8080,  // Servicio interno común
      8443   // Servicio interno común
    ];

    if (BLOCKED_PORTS.includes(port)) {
      return res.status(403).json({ error: 'El acceso a este puerto está bloqueado
    }

    const response = await axios.get(parsedUrl.toString(), {
      timeout: 5000,
      maxContentLength: 1 * 1024 * 1024, // máximo 1MB de contenido
      maxRedirects: 2,
      headers: {
```

```
        'User-Agent': 'MCP-SafeFetcher/1.0' // Identificar como servicio MCP
      }
    });

    // Sanitizar respuesta antes de devolver
    const contentLength = response.data.length;
    res.json({
      success: true,
      contentLength,
      preview: typeof response.data === 'string'
        ? response.data.substring(0, 500)
        : 'Contenido binario'
    });

  } catch (error) {
    console.error('Error de obtención:', error);
    res.status(400).json({ error: 'No se puede obtener contenido' });
  }
});

// Endpoint seguro: entrega de webhook con destino validado
app.post('/api/send-webhook', async (req, res) => {
  const { webhookUrl, payload } = req.body;

  if (!webhookUrl || typeof webhookUrl !== 'string') {
    return res.status(400).json({ error: 'webhookUrl inválido' });
  }

  if (!payload || typeof payload !== 'object') {
    return res.status(400).json({ error: 'Payload inválido' });
  }

  try {
    // Validar URL del webhook
    const parsedUrl = await validateUrl(webhookUrl, false);

    // Verificar que es un destino de webhook válido
    const hostname = parsedUrl.hostname;
    if (hostname === 'localhost' || hostname === '127.0.0.1') {
      return res.status(403).json({ error: 'No se puede enviar webhook a localhost
    }

    // Realizar solicitud de webhook con parámetros estrictos
    const response = await axios.post(
      parsedUrl.toString(),
      payload,
      {
        timeout: 10000,
```

```
        maxRedirects: 0, // Sin redirecciones para webhooks
        headers: {
          'Content-Type': 'application/json',
          'User-Agent': 'MCP-Webhook/1.0'
        }
      }
    );

    res.json({
      success: true,
      statusCode: response.status
    });

  } catch (error) {
    console.error('Error de webhook:', error);
    res.status(400).json({ error: 'Entrega de webhook fallida' });
  }
});

app.listen(3000);
```

1.8 Pasos de Remediación

1. **Implementar Validación y Filtrado de URL**: Crear una función robusta de validación de URL que verifique el protocolo (solo HTTP/HTTPS), realice resolución DNS con timeout, valide IPs resueltas contra rangos bloqueados (127.0.0.0/8, 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 169.254.0.0/16, etc.) y aplique límites de longitud razonable. Usar lista blanca para operaciones sensibles (proxy de imagen) y greylist/blacklist para operaciones generales.

2. **Bloquear Rangos de IP Internos y Servicios de Metadatos**: Bloquear explícitamente rangos de IP privados, direcciones loopback y endpoints de metadatos del proveedor de la nube (AWS: 169.254.169.254, GCP: metadata.google.internal, Azure: 169.254.169.254). Implementar protección contra DNS rebinding resolviendo nuevamente los hostnames antes de hacer solicitudes. Restringir el tráfico saliente a puertos específicos (bloquear 22, 3306, 5432, 6379, 27017, 9200, etc.).

3. **Implementar Limitación de Tasa y Aislamiento de Solicitudes**: Limitar el número de solicitudes por usuario/IP por período de tiempo. Usar namespaces de red separados o contenedores para hacer solicitudes salientes. Implementar timeouts de solicitud (5-10 segundos) y límites de tamaño estrictos (1-10MB dependiendo del caso de uso). Registrar todas las solicitudes salientes para auditoría e investigación.

4. **Aplicar Defensa en Profundidad con Monitoreo**: Usar políticas de red para restringir el acceso saliente a destinos específicos. Implementar Firewalls de Aplicación Web (WAF) con reglas de detección de SSRF. Monitorear patrones de DNS inusuales, múltiples intentos de conexión fallidos o solicitudes a rangos internos.

Realizar revisiones de seguridad regulares del código que maneja URLs suministradas por usuarios. Usar herramientas como Burp Suite para SSRF hunting.

1.9 Referencias

- OWASP Server-Side Request Forgery (SSRF)
- CWE-918: Falsificación de Solicitud del Lado del Servidor
- OWASP SSRF Prevention Cheat Sheet
- PortSwigger: Server-Side Request Forgery
- Seguridad de AWS: Mitigación de Ataques SSRF

Última Actualización: Enero 2026
Estado: Publicado
Idioma: Español