

Contents

1 SEC-011: Denegación de Servicio por Expresión Regular (ReDoS)	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	3
1.8 Pasos de Remediación	5
1.9 Referencias	6

1 SEC-011: Denegación de Servicio por Expresión Regular (ReDoS)

1.1 Severidad

Media □

1.2 Referencia OWASP

A04:2021 - Diseño Inseguro

1.3 Referencias CWE

- CWE-1333: Complejidad de Expresión Regular Ineficiente

1.4 Descripción

Las vulnerabilidades de Denegación de Servicio por Expresión Regular (ReDoS) ocurren cuando una aplicación usa expresiones regulares ineficientes que pueden causar retroceso catastrófico al procesar entrada maliciosa. En servidores MCP, los atacantes pueden proporcionar cadenas especialmente elaboradas que causan que el matching de regex consuma CPU excesiva, llevando a cuelgue del servidor, degradación del rendimiento o denegación de servicio. Esto es particularmente peligroso cuando los patrones regex se aplican a entrada controlada por el usuario sin salvaguardas.

1.5 Mapeo de Cumplimiento

1.5.1 SOC2

- **CC6.6:** Operaciones del Sistema - La organización previene ataques de denegación de servicio mediante validación eficiente de entrada.
- **CC6.8:** Operaciones del Sistema - El consumo de recursos se monitorea para detectar anomalías.

- **CC7.2:** Monitoreo del Sistema - La organización monitorea uso de CPU y detecta anomalías de rendimiento.

1.5.2 HIPAA

- **§164.312(a)(1):** Control de Acceso - Implementar validación segura de entrada para prevenir ataques DoS en sistemas de ePHI.
- **§164.312(b):** Controles de Auditoría - Monitorear patrones de rendimiento inusuales indicando ataques.

1.5.3 PCI DSS

- **6.5.1:** Las fallas de inyección incluyendo ReDoS se previenen mediante codificación segura
- **6.2.4:** Todo código personalizado se desarrolla con validación eficiente de entrada
- **11.3:** Las pruebas de seguridad incluyen pruebas para vulnerabilidades ReDoS

1.6 Ejemplo de Código Vulnerable

```
// ⚠ INSEGURO: Patrones regex vulnerables
const express = require('express');
const app = express();

app.use(express.json());

// Endpoint vulnerable: validación de email con retroceso catastrófico
app.post('/api/validate-email', (req, res) => {
  const { email } = req.body;

  // PELIGRO: Esta regex es vulnerable a ReDoS
  // Patrón: (a+)+$ causa retroceso exponencial
  const emailRegex = /^(([a-z]|[a-z]{2})+)+@([a-z]+\.)+[a-z]{2,}$/i;

  // Entrada del atacante: 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaab' causa pico de CPU
  if (emailRegex.test(email)) {
    res.json({ valid: true });
  } else {
    res.json({ valid: false });
  }
});

// Endpoint vulnerable: validación de contraseña
app.post('/api/validate-password', (req, res) => {
  const { password } = req.body;

  // PELIGRO: (w+)* permite ataque ReDoS
```

```

const passwordRegex = /^(\\w+)*$/;

// Entrada del atacante: cadena larga de caracteres 'w' causa cuelgue
if (passwordRegex.test(password)) {
  res.json({ valid: true });
} else {
  res.json({ valid: false });
}
});

app.listen(3000);

```

1.7 Implementación Segura

```

// ☐ SEGURO: Patrones regex seguros y validación de entrada
const express = require('express');
const { check, validationResult } = require('express-validator');
const app = express();

app.use(express.json());

// Función auxiliar segura para validación de email
function isValidEmail(email) {
  // Limitar longitud de entrada para prevenir incluso que regex seguro procese ca
  if (!email || email.length > 254) {
    return false;
  }

  // Usar patrón de regex seguro (sin retroceso catastrófico)
  // 0 mejor aún, usar una librería como validator.js
  const emailRegex = /^[^\\s@]+@[^\\s@]+\\. [^\\s@]+$/;

  return emailRegex.test(email);
}

// Función auxiliar segura para validación de contraseña
function isValidPassword(password) {
  if (!password || password.length < 8 || password.length > 128) {
    return false;
  }

  // Verificar criterios específicos en lugar de regex compleja
  const hasUpperCase = /[A-Z]/.test(password);
  const hasLowerCase = /[a-z]/.test(password);
  const hasNumber = /[0-9]/.test(password);
  const hasSpecialChar = /[!@#$%^&*]/.test(password);

```

```

    return hasUpperCase && hasLowerCase && hasNumber && hasSpecialChar;
}

```

// Endpoint seguro: validación de email con timeout

```

app.post('/api/validate-email', [
  check('email')
    .trim()
    .notEmpty().withMessage('Email es requerido')
    .isLength({ max: 254 }).withMessage('Email demasiado largo')
    .custom((value) => {
      if (!isValidEmail(value)) {
        throw new Error('Formato de email inválido');
      }
      return true;
    })
], (req, res) => {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const { email } = req.body;
  res.json({ valid: true, email });
});

```

// Endpoint seguro: validación de contraseña con límites de entrada

```

app.post('/api/validate-password', [
  check('password')
    .isLength({ min: 8, max: 128 })
    .withMessage('Contraseña debe ser 8-128 caracteres')
    .custom((value) => {
      if (!isValidPassword(value)) {
        throw new Error('Contraseña debe contener mayúscula, minúscula, número y c
      }
      return true;
    })
], (req, res) => {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  res.json({ valid: true });
});

```

// Endpoint seguro: validación de URL con regex seguro

```

app.post('/api/validate-url', [
  check('url')
    .trim()
    .isLength({ max: 2048 })
    .withMessage('URL demasiado larga')
    .isURL()
    .withMessage('Formato de URL inválido')
], (req, res) => {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const { url } = req.body;
  res.json({ valid: true, url });
});

// Endpoint seguro: validación de dirección IP
app.post('/api/validate-ip', [
  check('ip')
    .trim()
    .isIP()
    .withMessage('Dirección IP inválida')
], (req, res) => {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const { ip } = req.body;
  res.json({ valid: true, ip });
});

app.listen(3000);

```

1.8 Pasos de Remediación

1. **Evitar Patrones Regex Complejos con Retroceso:** Usar patrones regex seguros sin cuantificadores anidados (p. ej., evitar (a+)+, (a)). Preferir patrones de regex lineales o usar funciones de librería (validator.js, is.js) en lugar de regex personalizado. Probar patrones regex con herramientas de detección ReDoS. Limitar longitud de entrada antes del procesamiento regex para reducir impacto.
2. **Usar Librerías de Validación de Entrada en Lugar de Regex Personalizado:** Usar librerías como express-validator, joi o yup que tienen patrones de validación seguros. Estas librerías han sido ampliamente probadas por vulner-

abilidades ReDoS. Validar formato de entrada, longitud y lista blanca de caracteres cuando sea posible en lugar de usar patrones complejos.

3. **Implementar Mecanismos de Timeout para Operaciones Regex:** Establecer timeouts de ejecución en matching de regex (Node.js no tiene timeouts de regex incorporados, pero herramientas externas pueden ayudar). Monitorear tiempo de ejecución de regex y alertar si patrones exceden duración esperada. Usar métodos de validación alternativos (lista blanca de caracteres, métodos de cadena) para validaciones simples.
4. **Probar y Monitorear para ReDoS:** Usar herramientas como OWASP ReDoS-Scanner o regex101.com para probar patrones por vulnerabilidades. Incluir pruebas ReDoS en procedimientos de pruebas de seguridad. Monitorear patrones de uso de CPU para picos durante validación de entrada. Implementar limitación de tasa en endpoints de validación para mitigar impacto de ataque.

1.9 Referencias

- [OWASP Regular Expression Denial of Service](#)
- [CWE-1333: Complejidad de Expresión Regular Ineficiente](#)
- [ReDoS Detection and Prevention Guide](#)
- [Safe Regex Testing Tool](#)
- [express-validator Documentation](#)

Última Actualización: Enero 2026

Estado: Publicado

Idioma: Español