Contents

1 SEC-010: Limitación de Tasa Faltante **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 . . . 4
1.9 Referencias . . . 5

1 SEC-010: Limitación de Tasa Faltante

1.1 Severidad

**Media **

1.2 Referencia OWASP

A04:2021 - Diseño Inseguro

1.3 Referencias CWE

- CWE-307: Restricción Impropia de Capas o Marcos de UI Renderizados
- CWE-770: Asignación de Recursos Sin Límites o Restricción

1.4 Descripción

Las vulnerabilidades de Limitación de Tasa Faltante ocurren cuando una aplicación no implementa mecanismos de throttling de solicitudes o limitación de tasa, permitiendo a los atacantes inundar el servidor con solicitudes. En servidores MCP, esto permite ataques de fuerza bruta, relleno de credenciales, abuso de API, ataques de denegación de servicio o agotamiento de recursos. Sin limitación de tasa, los atacantes pueden intentar autenticación ilimitada, operaciones de spam o agotar recursos del servidor.

1.5 Mapeo de Cumplimiento

1.5.1 SOC2

- **CC6.1**: Control de Acceso - La limitación de tasa previene intentos de acceso no autorizado.
- **CC6.8**: Operaciones del Sistema - La organización previene agotamiento de recursos mediante limitación de tasa.
- **CC7.2**: Monitoreo del Sistema - La organización monitorea y detecta patrones de tráfico anormal.

1.5.2 HIPAA

- **§164.312(b)**: Controles de Auditoría - Implementar monitoreo para patrones de acceso sospechosos e intentos de fuerza bruta.
- **§164.312(a)(1)**: Control de Acceso - La limitación de tasa protege contra ataques de fuerza bruta en acceso de ePHI.

1.5.3 PCI DSS

- **6.5.10**: Los ataques de fuerza bruta se previenen mediante limitación de tasa y bloqueo de cuenta
- **11.3**: Las pruebas regulares de seguridad incluyen pruebas de resistencia a fuerza bruta
- **10.2**: Todos los intentos de acceso incluyendo intentos fallidos deben ser monitoreados

1.6 Ejemplo de Código Vulnerable

```
// □ INSEGURO: Sin limitación de tasa en login
const express = require('express');
const app = express();

app.use(express.json());

// Endpoint vulnerable: intentos de login ilimitados
app.post('/api/login', async (req, res) => {
  const { username, password } = req.body;

  // Sin limitación de tasa - el atacante puede intentar credenciales ilimitadas
  const user = await authenticateUser(username, password);

  if (user) {
    res.json({ success: true, token: generateToken(user) });
  } else {
    res.status(401).json({ error: 'Credenciales inválidas' });
  }
});

// Endpoint vulnerable: sin límites en llamadas API
app.get('/api/data', (req, res) => {
  // El atacante puede hacer solicitudes ilimitadas
  const data = fetchData();
  res.json(data);
});

app.listen(3000);
```

1.7 Implementación Segura

```
// ✅ SEGURO: Limitación de tasa con express-rate-limit
const express = require('express');
const rateLimit = require('express-rate-limit');
const RedisStore = require('rate-limit-redis');
const redis = require('redis');
const app = express();

app.use(express.json());

// Crear cliente Redis para limitación de tasa distribuida
const client = redis.createClient({
  host: process.env.REDIS_HOST || 'localhost',
  port: process.env.REDIS_PORT || 6379
});

// Limitador estricto para login (5 intentos por 15 minutos)
const loginLimiter = rateLimit({
  store: new RedisStore({
    client: client,
    prefix: 'login-limit:'
  }),
  windowMs: 15 * 60 * 1000, // 15 minutos
  max: 5, // 5 solicitudes por windowMs
  message: 'Demasiados intentos de login, intenta más tarde',
  standardHeaders: true,
  legacyHeaders: false,
  skip: (req) => {
    // Omitir limitación de tasa para IPs de admin
    const adminIPs = process.env.ADMIN_IPS?.split(',') || [];
    return adminIPs.includes(req.ip);
  }
});

// Limitador API general (100 solicitudes por 15 minutos)
const generalLimiter = rateLimit({
  store: new RedisStore({
    client: client,
    prefix: 'general-limit:'
  }),
  windowMs: 15 * 60 * 1000,
  max: 100,
  standardHeaders: true,
  legacyHeaders: false
});

// Endpoint seguro: login con limitación de tasa
```

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

  try {
    const user = await authenticateUser(username, password);

    if (user) {
      // En login exitoso, opcionalmente reiniciar el límite de tasa
      res.json({ success: true, token: generateToken(user) });
    } else {
      res.status(401).json({ error: 'Credenciales inválidas' });
    }
  } catch (error) {
    res.status(500).json({ error: 'Autenticación fallida' });
  }
});

// Endpoint seguro: datos API con limitación de tasa general
app.get('/api/data', generalLimiter, (req, res) => {
  const data = fetchData();
  res.json(data);
});

app.listen(3000);
```

1.8 Pasos de Remediación

1. **Implementar Limitación de Tasa en Todos los Endpoints**: Aplicar limitación estricta en endpoints de autenticación (5-10 solicitudes por 15 minutos). Usar límites moderados para endpoints API generales (100-1000 por hora). Usar limitación de tasa distribuida (Redis/Memcached) para despliegues multi-servidor. Rastrear límites de tasa por dirección IP, ID de usuario y clave API.

2. **Usar Backoff Exponencial para Reintentos**: Implementar retrasos progresivos después de intentos fallidos. Bloquear cuentas temporalmente después de múltiples intentos de login fallidos. Proporcionar retroalimentación clara a usuarios sobre estado de límite de tasa. Implementar aumento gradual de límite de tasa para usuarios legítimos en el tiempo.

3. **Monitorear y Alertar sobre Violaciones de Límite de Tasa**: Registrar todas las violaciones de límite de tasa para análisis de seguridad. Alertar a equipos de seguridad sobre patrones sospechosos (múltiples IPs alcanzando límites, comportamiento tipo bot). Implementar CAPTCHA después de fallos repetidos. Usar detección de anomalías para identificar ataques coordinados.

4. **Implementar Limitación de Tasa Escalonada**: Diferentes límites para usuarios autenticados vs. no autenticados. Límites más altos para usuarios premium/confiables. Límites más bajos para usuarios nuevos/no confiables. Lista blanca de IPs críticas (sistemas de monitoreo, servicios internos).

1.9 Referencias

- OWASP Rate Limiting Cheat Sheet
- CWE-770: Asignación de Recursos Sin Límites
- express-rate-limit Documentation

Última Actualización: Enero 2026
Estado: Publicado
Idioma: Español