# Relatório de Pentest

## Ricado Amorim - Pentest Black Box (Mobile)

Período: 31/08/2025 até 29/09/2025

| 0 | 1 | 0 | 0 |
|---|---|---|---|
| Críticas | Altas | Médias | Baixas |

## Vulnerabilidades Detalhadas

## 1. Rate Limit

HIGH

**CVSS:** 8

**Status:** open

**Data de Descoberta:** 26/09/2025

**Complexidade de Mitigação:** medium

### Descrição:

# Rate Limit - Ausência de Limitação de Taxa

**Severidade:** Crítica | **CVSS:** 9.8

## Resumo Executivo

A vulnerabilidade de Rate Limit ocorre quando não há restrição apropriada no número de solicitações que um usuário ou cliente pode realizar em um determinado período de tempo. Isso permite que um invasor abuse de funcionalidades como APIs públicas, formulários de login ou endpoints sensíveis, resultando em ataques de força bruta, exfiltração de dados ou negação de serviço.

## Explicação Técnica

Quando um sistema não implementa controles adequados de **Rate Limiting**, ele se torna suscetível a explorações onde um atacante pode realizar um número ilimitado de requisições em um curto período. Essa falha geralmente pode ser explorada em APIs RESTful, formulários web ou qualquer interface que aceite interações repetidas de usuários.

Os ataques exploram a ausência de mecanismos como:

- Limitação de requisições por IP ou chave de API.
- Uso de tokens temporários para validação de requisições.
- Implementação de captchas para limitar interações automáticas.

Essa vulnerabilidade é particularmente perigosa em endpoints que executam operações críticas, como autenticação, recuperação de dados ou transações financeiras.

## Métodos de Exploração

1. **Força Bruta**: Tentativas repetidas de autenticação para descobrir credenciais válidas.
2. **Enumeração de Dados**: Extração massiva de informações, como consulta de dados públicos ou privados.

3. **Denial of Service (DoS)**: Grande volume de requisições para sobrecarregar o sistema.

## Impacto e Consequências

A ausência de Rate Limiting pode levar a:

- **Comprometimento de contas** devido a ataques de força bruta.
- **Exfiltração de dados sensíveis** através de consultas massivas.
- **Interrupção do serviço** por sobrecarga de recursos.

Esses impactos podem comprometer a integridade, confidencialidade e disponibilidade do sistema.

## Exemplos Práticos

### Exemplo 1: Força Bruta em Formulário de Login

```
import requests

url = "https://example.com/login"
for i in range(1000000):
    data = {"username": "admin", "password": f"password{i}"}
    response = requests.post(url, data=data)
    if "Login Successful" in response.text:
        print(f"Senha descoberta: password{i}")
        break
```

### Exemplo 2: Enumeração de Dados em API

```
import requests

url = "https://example.com/api/data"
for i in range(1, 10000):
    response = requests.get(f"{url}/{i}")
    if response.status_code == 200:
        print(response.json())
```

## Cenários de Ataque

- **Autenticação**: Um atacante utiliza um script automatizado para tentar milhões de combinações de senha em um endpoint de login sem restrições de requisições.
- **API Pública**: Um invasor explora uma API pública para baixar grandes volumes de dados, impactando o desempenho do serviço e comprometendo informações.

- **DoS**: Um atacante gera tráfego massivo para um endpoint, causando indisponibilidade do sistema.

## Detecção e Identificação

Para identificar a ausência de **Rate Limiting**, é recomendável realizar os seguintes testes:

- Enviar múltiplas requisições consecutivas para o mesmo endpoint e observar a resposta.
- Monitorar logs para detectar padrões de uso

### Remediação:

# Plano de Remediação - Rate Limit

**Status:** Implementação Imediata | **Prioridade:** Crítica

## Resumo da Correção

A vulnerabilidade de tipo **Rate Limit** ocorre quando uma aplicação não possui mecanismos adequados para limitar o número de requisições feitas em um intervalo de tempo, permitindo abusos como ataques de força bruta ou negação de serviço (DoS). A correção envolve a implementação de mecanismos de controle de taxa, autenticação e monitoramento.

## Passos de Implementação

1. **Identificar os endpoints vulneráveis**
2. Realizar uma auditoria em todos os endpoints da aplicação para identificar quais estão sem proteção contra excesso de requisições.
3. **Implementar um middleware de rate limiting**
4. Utilizar bibliotecas específicas para implementar limites de requisições por IP ou usuário, como `express-rate-limit` no Node.js ou `django-ratelimit` no Django.
5. **Configurar limites adequados**
6. Definir limites de requisição com base no uso esperado do sistema, como 100 requisições por minuto por IP ou por token de autenticação.
7. **Adicionar mensagens de erro adequadas**
8. Retornar códigos de status HTTP como `429 Too Many Requests` quando os limites forem excedidos, com uma mensagem clara no corpo da resposta.
9. **Implementar mecanismos de bloqueio temporário**

10. Bloquear temporariamente o acesso a usuários ou IPs que excederem o limite repetidamente dentro de um intervalo de tempo.

## Código Corrigido

### Exemplo em Node.js

### Antes (Vulnerável):

```
app.get('/api/resource', (req, res) => {
  res.send('Recurso acessado');
});
```

### Depois (Seguro):

```
const rateLimit = require('express-rate-limit');

const limiter = rateLimit({
  windowMs: 1 * 60 * 1000, // 1 minuto
  max: 100, // Limite de 100 requisições por minuto
  message: 'Muitas requisições enviadas. Tente novamente mais tarde.',
  headers: true,
});

app.use('/api/resource', limiter);

app.get('/api/resource', (req, res) => {
  res.send('Recurso acessado');
});
```

## Configurações de Segurança

- Habilitar logs detalhados para monitoramento de requisições e detecção de abusos.
- Configurar o WAF (Web Application Firewall) para bloquear IPs maliciosos automaticamente.
- Utilizar autenticação baseada em tokens (JWT, OAuth) para controlar acessos.

## Validação e Testes

- Realizar testes de carga simulando múltiplas requisições simultâneas para verificar o funcionamento do rate limiting.
- Executar testes de penetração específicos para força bruta e DoS em endpoints sensíveis.

- Validar mensagens de erro para verificar se o código `429` é retornado corretamente.

## Medidas Preventivas

- Documentar os limites de requisição e compartilhá-los com desenvolvedores e usuários da API.
- Implementar autenticação robusta para endpoints críticos.
- Educar a equipe de desenvolvimento sobre a importância de mecanismos de rate limiting em aplicações web.

## Monitoramento Contínuo

### Dica Técnica: