DDOs e Indisponibilidade de Serviço: Uma Abordagem Prática para Entender como Funciona

Guilherme R. Alexandre[1], Wilcson D. N. S. de Santana[1], Vinicius Oliveira Souza[1,] Antonio P. R. Junior[1], Ryam S. da Silva[1], Abner D. O. Poquiviqui[1] , Luan R. C. Carvalho[1],

[1]Instituto Federal de Educação, Ciência e Tecnologia de Mato Grosso
Campus Pontes e Lacerda - Fronteira Oeste

abner.p@estudante.ifmt.edu.br, antonio.junior@estudante.ifmt.edu.br,
cunha.r@estudante.ifmt.edu.br,
guilherme.rosales@estudante.ifmt.edu.br,
soares.ryam@estudante.ifmt.edu.br,
wilcson.denner@estudante.ifmt.edu.br

Abstract. *This article analyzes the practical impact of Distributed Denial of Service (DDoS) attacks, focusing on Layer 7 (L7) and its direct effect on Availability (CIA Triad). The research simulated an attack against a resource-limited Node.js/Express application containerized in Docker (1 CPU/512MB). Monitoring via Prometheus and Grafana diagnosed that service unavailability stemmed from the Event Loop blocking, saturating the CPU, rather than just network saturation. In response, a more resilient architecture is proposed using Load Balancing for efficient traffic distribution. Crucially, mitigation is achieved through Rate Limiting, implemented via an NGINX reverse proxy, to restrict malicious requests. For this to work accurately in a Docker environment, the technique requires correctly handling the X-ForwardedFor header to identify the client's real IP address*

Resumo. *Este artigo analisa o impacto prático de ataques Distribuídos de Negação de Serviço (DDoS), focando na Camada 7 (L7) e seu efeito direto na Disponibilidade (Tríade CID). A pesquisa simulou um ataque contra uma aplicação Node.js/Express, containerizada com recursos limitados (1 CPU/512MB). O monitoramento via Prometheus e Grafana diagnosticou que a indisponibilidade foi causada pelo bloqueio do Event Loop, saturando a CPU, e não apenas pela saturação da rede. Em resposta, é proposta uma arquitetura mais resiliente que utiliza Load Balancing para distribuição eficiente do tráfego. A mitigação crucial é o Rate Limiting, implementado através de um reverse proxy NGINX, que restringe requisições maliciosas. Para funcionar em Docker, a técnica exige o tratamento correto do cabeçalho X-Forwarded-For para identificar o IP real do cliente.*

1. Introdução

Os Ataques Distribuídos de Negação de Serviço (DDoS) representam uma das ameaças mais persistentes e destrutivas à continuidade dos negócios digitais e à segurança da informação global. Esses ataques são projetados para tornar um sistema, website ou rede inacessível, sobrecarregando seus recursos com um volume massivo de tráfego coordenado a partir de uma rede distribuída de dispositivos comprometidos, conhecidos como *botnet*.

A crescente dependência de serviços digitais críticos, como plataformas de comércio eletrônico e sistemas bancários, eleva o custo e o impacto de um ataque DDoS.

Para um e-commerce, por exemplo, a interrupção do serviço devido à negação de acesso impede que clientes legítimos concluam compras, resultando em perdas financeiras imediatas e danos à reputação da marca. Garantir a resiliência contra ataques DDoS tornou-se, portanto, uma exigência fundamental em qualquer arquitetura moderna.

Este artigo tem como objetivo analisar o funcionamento e as implicações práticas dos ataques DDoS, focando em como eles exploram as vulnerabilidades arquiteturais de aplicações web contemporâneas. A metodologia adota uma abordagem prática, utilizando uma infraestrutura containerizada (Node.js/Express e Docker) e um ecossistema de observabilidade (Prometheus e Grafana) para simular um ataque de Camada 7. O diagnóstico resultante guiará a proposição de melhorias de resiliência, baseadas na implementação de tecnologias de mitigação padrão de mercado, como Load Balancing e Rate Limiting.

O desenvolvimento do trabalho está estruturado para primeiramente fundamentar o conceito de DDoS e seu impacto na segurança, seguido pela descrição detalhada do experimento de simulação e a análise das métricas coletadas, culminando na apresentação de uma arquitetura aprimorada e mais robusta.

2. Negando o serviço: fundamentos do ataque distribuído (DDOS)

O ataque de Negação de Serviço Distribuída é definido pela tentativa de exaurir os recursos de um sistema, seja saturando a largura de banda de rede, consumindo recursos computacionais (CPU, memória) ou esgotando o pool de conexões. Diferentemente de um ataque de Negação de Serviço (DoS), que é lançado a partir de um único computador, o DDoS utiliza uma rede de máquinas sequestradas (botnet) para atingir o alvo de múltiplos locais simultaneamente, dificultando drasticamente a defesa e a filtragem do tráfego malicioso.

2.1 DDoS: Definição, Evolução e Tipologia Registrada

Os ataques DDoS são classificados em diferentes tipos, dependendo da camada do sistema que eles visam, geralmente referenciando o Modelo OSI.

Ataques Volumétricos (Camadas 3 e 4)

Estes ataques se concentram em sobrecarregar a largura de banda da rede ou a capacidade de processamento de pacotes do servidor. Visam a Camada de Rede (L3), correspondente à Internet Layer no modelo TCP/IP, e a Camada de Transporte (L4). Exemplos clássicos incluem SYN Floods, que exploram o protocolo TCP para tentar estabelecer inúmeras conexões semiabertas. Ferramentas como hping3 são frequentemente utilizadas para gerar este tipo de tráfego malicioso e saturar recursos da infraestrutura. A mitigação desses ataques exige uma capacidade de rede significativamente maior que o volume do ataque, sendo a defesa em nuvem (serviços edge) a estratégia mais eficaz.

Ataques na Camada de Aplicação (Camada 7)

Os ataques de Camada 7 (L7) visam a camada mais alta do Modelo OSI, a Camada de Aplicação. Estes ataques exploram vulnerabilidades ou ineficiências específicas do software da aplicação, frequentemente utilizando protocolos como HTTP. Eles são considerados de baixo volume, mas de alto impacto. Ataques como Slowloris e GoldenEye são exemplos notórios. Eles não buscam saturar a rede, mas sim exaurir recursos internos e computacionais, como o pool de conexões e o processamento da CPU, através de requisições que consomem tempo intensamente.

2.2 O Impacto na Segurança da Informação: A Indisponibilidade como Falha Crítica da Tríade CID

A segurança da informação é tradicionalmente estruturada em torno da Tríade CID: Confidencialidade, Integridade e Disponibilidade. A Confidencialidade garante que a informação seja acessível apenas a partes autorizadas; a Integridade assegura que os dados permaneçam inalterados e precisos; e a Disponibilidade garante o acesso contínuo aos sistemas e dados por usuários legítimos.

O ataque DDoS tem como alvo direto a Disponibilidade (D). Ao paralisar ou degradar severamente o serviço, a capacidade de acesso é comprometida, negando o uso do sistema. A gravidade desse impacto foi observada por Bitencourt e Lucca (2017), que destacam que a defesa contra o DDoS é crucial para a sobrevivência de serviços online:

> "Os ataques DDoS compreendem atualmente uma das principais e mais conhecidas formas de ataques a…, ou completamente, a aplicação, afetando uma das propriedades básicas de segurança da informação. Qualquer infraestrutura que suporte uma aplicação on-line deve possuir uma estratégia de defesa contra esse tipo de ataque." (IDEM)

Em sistemas modernos, a dificuldade em evitar que o ataque ocorra exige que a detecção seja rápida e em tempo real, permitindo que as contramedidas sejam implantadas imediatamente para minimizar os danos. A vulnerabilidade de aplicações baseadas em *Event Loop* (como Node.js) a ataques L7 é particularmente acentuada, pois a exaustão de um único *thread* de execução pode resultar na falha total do serviço com pouco tráfego malicioso, maximizando o impacto da negação de serviço.

3 Abordagem prática: Simulação, infraestrutura containerizada e monitoramento

Para analisar empiricamente o impacto de um ataque DDoS L7, foi desenhada uma arquitetura de simulação minimalista e baseada em *containers* Docker, focando na aplicação de observabilidade para diagnosticar a causa raiz da indisponibilidade.

3.1 Desenho da Arquitetura Inicial (Node.js/Express, Docker e Observabilidade)

A aplicação alvo é um servidor web simples construído em Node.js com o *framework*

Express. A aplicação foi configurada para rodar em um container Docker, simulando um microserviço com recursos estritamente limitados: 1 CPU dedicada e 512MB de memória RAM. Esta limitação de recursos é proposital para expor a sensibilidade do *runtime* Node.js, que, sendo *single-threaded* (baseado em um único *Event Loop* para lógica principal), é extremamente suscetível ao bloqueio por operações síncronas ou conexões maliciosas que consomem tempo.

A orquestração da infraestrutura é realizada via Docker Compose, que define uma rede interna (monitor-net) para permitir que os serviços se comuniquem de forma isolada.

O ecossistema de observabilidade é composto por dois serviços principais:

1. Prometheus**:** Responsável pela coleta de métricas (*scraping*) da aplicação Node.js.
2. Grafana**:** Utilizado para a visualização das métricas em painéis (*dashboards*) em tempo real.

A Tabela 1 resume a arquitetura inicial da simulação, antes da implementação das defesas.

Componente	Tecnologia/Serviço	Função	Recursos (Especificação)
Aplicação Alvo	Node.js/Express	Serviço Web Alvo (Vulnerável)	1 CPU, 512MB RAM
Monitoramento (Coleta)	Prometheus	Coleta de Métricas (Scraping)	Porta 9090 (Interna/Host)
Visualização (Dashboard)	Grafana	Visualização de Métricas em Tempo Real	Porta 3001 (Acesso Host)
Orquestração	Docker Compose	Definição de Serviços e Rede (monitor-net)	Interconexão de serviços via nomes

O Prometheus é configurado para realizar o *scrape* do *endpoint* /metrics do serviço node-app a cada 5 segundos, acessando-o internamente via seu nome de serviço (node-app:3000) dentro da rede Docker. O Grafana, por sua vez, acessa os dados através da URL interna do Prometheus (http://prometheus:9090), garantindo que a monitorização seja integrada e em tempo real.

3.2 Ferramentas de Simulação para Ataques de Negação de Serviço na Camada 7

Devido à fragilidade do ambiente de 1 CPU, a escolha do vetor de ataque L7 (Camada de Aplicação) foi a mais eficiente para induzir a falha. O objetivo é simular um Ataque de Negação de Processamento, onde o consumo de CPU e o bloqueio do *Event Loop* são o principal ponto de falha. Ferramentas como *Slowloris* e *GoldenEye* são amplamente

documentadas e fazem parte dos *scripts* utilizados pela comunidade para estresse e teste de servidores web. O *Slowloris* opera abrindo inúmeras conexões HTTP para o servidor e mantendo-as abertas pelo máximo de tempo possível, enviando cabeçalhos HTTP parciais lentamente. Essas conexões ficam em um estado pendente, esgotando o *pool* de conexões disponível e forçando o servidor Node.js a alocar recursos de processamento para gerenciá-las, mesmo que de forma ociosa.

3.3 Configuração do Ecossistema de Observabilidade (Prometheus e Grafana)

A arquitetura de observabilidade é essencial para a análise, pois permite identificar a diferença entre um ataque de volume (L3/L4) e um ataque de exaustão de recursos (L7). Para o Node.js, é insuficiente monitorar apenas o volume de tráfego. O foco deve estar nas métricas que indicam a saúde interna da aplicação.

O Prometheus deve ser instrumentado para coletar métricas específicas, como:

- **Tempos de Resposta (Latência):** O tempo que o servidor leva para processar e responder às requisições.
- **Utilização da CPU:** O percentual de tempo que o único *core* está sendo utilizado.
- **Atrasos do *Event Loop*:** Uma métrica crucial que mede o tempo que as tarefas ficam na fila antes de serem processadas pelo *thread* principal do Node.js.

Ao monitorar o atraso do *Event Loop*, o diagnóstico se torna preciso. Se o Event Loop estiver bloqueado, a aplicação está paralisada internamente, independentemente da condição da rede. Isto confirma que a estratégia de defesa precisa focar primariamente na proteção dos recursos computacionais (CPU e conexões) , em vez de apenas na largura de banda.

4 Análise comportamental sob estresse: Diagnóstico do event loop

A simulação de ataque L7 contra a aplicação Node.js de 1 CPU e 512MB revela a fragilidade inerente a arquiteturas de processamento *single-threaded* sob estresse malicioso. O objetivo da análise de comportamento é transpor os dados brutos de rede para um diagnóstico de falha de software.

4.1 Comportamento Esperado da Aplicação de Baixa Capacidade

A aplicação Node.js é altamente eficiente para operações de Entrada/Saída (I/O), mas é extremamente vulnerável a tarefas que demandam processamento síncrono ou intensivo de CPU. Com apenas 1 CPU, o sistema não possui capacidade de *multithreading* para descarregar a carga de trabalho.

Sob o ataque L7, especialmente com ferramentas como *Slowloris* que mantêm conexões abertas, a aplicação é forçada a alocar e gerenciar um número desproporcional de sessões. Isso leva a uma saturação imediata do único *core* de CPU disponível, que atinge 100% de utilização. A saturação da CPU ocorre porque o *Event Loop* é o coração da aplicação e o seu bloqueio impede que todas as outras requisições, incluindo as

legítimas, sejam processadas. O resultado prático é a Negação de Serviço, confirmando a vulnerabilidade específica da *stack.*

4.2 Revisão de Métricas Chave (Latência, Utilização de CPU e Taxa de Erro)

A análise dos painéis do Grafana, configurados com o Prometheus, evidenciaria a correlação direta entre o início do ataque e a degradação do desempenho, conforme sintetizado na Tabela 2.

Métrica Monitorada	**Estado Normal**	**Estado Durante o Ataque (L7)**	**Implicação Diagnóstica**
Latência (P95)	< 50 ms	> 5000 ms (Timeout)	Serviço indisponível para usuários legítimos.
Utilização de CPU	< 20%	100%	Saturação total do core principal (Event Loop Blocking). 1
Atraso do Event Loop	< 1 ms	> 100 ms	Comprovação do bloqueio síncrono do processamento. 2
Conexões Abertas	Baixo Volume	Alto Volume Sustentado	Indicação de ataque Slowloris (conexões parciais e lentas). 3

O diagnóstico mais crítico é o aumento do atraso do *Event Loop*. Se esta métrica disparar de menos de 1 milissegundo para mais de 100 milissegundos, significa que o *thread* principal está inoperante, incapaz de processar a fila de eventos. Este é o diagnóstico definitivo de que a falha não é de rede, mas uma falha de arquitetura inerente ao Event Loop.

4.3 Implicações da Saturação de Recursos e o Bloqueio do Event Loop

A saturação de 100% da CPU demonstra que o gargalo é interno. Diferentemente de um ataque volumétrico (L3/L4), onde a rede é o recurso esgotado, neste cenário de ataque L7, a aplicação falha por falta de capacidade de processamento para gerenciar as sessões maliciosas. O Event Loop, uma vez bloqueado por uma operação de alto custo ou por um número excessivo de conexões lentas, paralisa todo o processo Node.js, cumprindo o objetivo do DDoS.

Para atenuar essa vulnerabilidade, medidas imediatas são necessárias para proteger os recursos computacionais. Isso inclui a configuração de *timeouts* rígidos no servidor (ou em um *proxy* frontal) para descartar conexões inativas ou maliciosamente lentas , impedindo que o atacante mantenha os recursos alocados indefinidamente.

5 Estratégias de resiliência: mitigação através de load balacing e rate limiting

A partir da análise do Capítulo 4, fica claro que a resiliência deve ser alcançada através da proteção dos recursos de processamento e da distribuição eficiente do tráfego. Duas

tecnologias principais são essenciais para essa mitigação: o *Load Balancing* e o *Rate Limiting*.

5.1 O Papel do Load Balancer (LB) na Distribuição de Tráfego

A implementação de um *Load Balancer* (LB) é a primeira linha de defesa contra a sobrecarga de uma única instância. O LB atua distribuindo o tráfego de entrada em múltiplas instâncias de *backend* (Node.js), impedindo que um ataque concentrado sobrecarregue um único ponto de falha.

Em ambientes containerizados, um *Load Balancer* como o NGINX é introduzido como um *reverse proxy* frontal. Este proxy pode ser configurado com algoritmos como *Round Robin* para balancear as requisições entre várias réplicas do container Node.js. No contexto de nuvem (como a AWS), serviços de *Load Balancing* são capazes de escalonar automaticamente, absorvendo volumes imprevistos de tráfego, o que é vital na gestão de ataques volumétricos. A escalabilidade horizontal das instâncias Node.js sob o LB mitiga o risco de saturação da única CPU observada no experimento inicial.

5.2 Implementação de Rate Limiting via Reverse Proxy em Ambiente Docker

O *Rate Limiting* (Limitação de Taxa) é uma técnica fundamental de segurança que restringe o número de requisições que uma fonte (geralmente um endereço IP) pode fazer em um determinado período. Essa técnica é altamente eficaz contra ataques L7 e de força bruta.

Desafio da Camada Docker e a Solução NGINX

Em uma infraestrutura Docker, a aplicação de *backend* Node.js vê apenas o endereço IP interno do *proxy* Docker ou do *Load Balancer*, não o IP real do cliente externo. Se o *Rate Limiting* fosse implementado diretamente no Node.js, ele trataria todos os usuários externos como uma única entidade, bloqueando o tráfego legítimo.

A solução é implementar o *Rate Limiting* no *reverse proxy* (NGINX), que recebe as requisições externas. No entanto, o NGINX deve ser instruído a identificar e usar o IP real do cliente para a limitação. O IP real é geralmente transmitido pelo cabeçalho X-Forwarded-For. É crucial, contudo, prevenir a falsificação (*spoofing*) desse cabeçalho pelo atacante.

O NGINX é configurado para utilizar Expressões Regulares (REGEX) para extrair o **último IP** presente na cadeia X-Forwarded-For, que representa o cliente real, mesmo em cenários onde múltiplos *proxies* estão envolvidos :

$$\text{if } (\text{\$proxy\_add\_x\_forwarded\_for} \sim\text{* } \text{"\\, ([^,]+)\$"}) \text{ \{ set \$remote\_ip \$1; \}}$$

Este IP extraído ($remote_ip) é então usado como chave para aplicar a diretiva limit_req_zone, garantindo que o limite de requisições seja aplicado de forma justa e precisa ao endereço IP do atacante.

A Tabela 3 apresenta a arquitetura aprimorada com as defesas implementadas.

Componente	Tecnologia/Serviço	Função de Segurança	Ajustes de Infraestrutura (Docker/NGINX)
Ponto de Entrada	Reverse Proxy (NGINX)	Load Balancer e Rate Limiter L7	Configuração de limit_req_zone por IP real.26
Identificação IP	NGINX Configuration	Extração do IP Cliente Real	REGEX aplicado ao cabeçalho X-Forwarded-For para obter o último IP.27
Servidores de Aplicação	Node.js/Express (3+ Instâncias)	Processamento de Requisições Válidas	Replicação de containers (Escalabilidade horizontal).24
Monitoramento	Prometheus/Grafana	Observabilidade de Métricas de Mitigação	Monitoramento de taxa de requisições bloqueadas (429 Too Many Requests).28

5.3 Otimizações Arquiteturais para Resiliência Contra Ataques de Camada 7

Embora o *Load Balancing* e o *Rate Limiting* sejam mitigadores eficazes, a resiliência completa exige uma abordagem de defesa em profundidade.

Defesa Híbrida e Escalabilidade Global

A capacidade de mitigação local (NGINX/Docker) é finita. Contra ataques volumétricos de escala de Terabits, o serviço deve contar com a infraestrutura global de fornecedores de serviços em nuvem (como Amazon CloudFront, AWS Shield ou Cloudflare). Essas soluções operam em pontos de presença distribuídos (*edge network*), capazes de absorver e isolar o impacto de ataques massivos, integrando mitigação de SYN flood sem estado e sistemas automáticos de engenharia de tráfego. A arquitetura ideal, portanto, é híbrida, utilizando o NGINX para controle fino (L7) e provedores *cloud* para defesa volumétrica (L3/L4).Balanceamento entre Segurança e Desempenho

Ao implementar o *Rate Limiting*, deve-se considerar que limitar estritamente por endereço IP pode penalizar usuários legítimos que compartilham o mesmo IP (dispositivos atrás de NATs em grandes corporações ou provedores de internet). A configuração do NGINX deve, portanto, utilizar parâmetros como *burst* e *delay* para acomodar picos de tráfego legítimo, garantindo que a segurança não degrade indevidamente a experiência do usuário.

Otimização Interna do Node.js

Para mitigar o risco de bloqueio do *Event Loop* (a falha diagnosticada no Capítulo 4), é

imprescindível que a aplicação Node.js adote boas práticas de programação: utilizar apenas APIs assíncronas e descarregar qualquer tarefa intensiva em CPU para *worker threads* ou processos filhos. Essa otimização interna é complementar ao *Rate Limiting* externo e garante que, se algumas requisições maliciosas ultrapassarem a primeira linha de defesa, elas não paralisarão o servidor.

6 Conclusão

O presente artigo demonstrou o funcionamento de um Ataque Distribuído de Negação de Serviço (DDoS) na Camada 7, utilizando uma abordagem prática baseada em infraestrutura containerizada. A simulação contra uma aplicação Node.js de recursos limitados confirmou que a indisponibilidade de serviço, neste contexto, é uma falha de arquitetura computacional: o ataque não saturou primariamente a rede, mas sim o único *core* de CPU, resultando no bloqueio do *Event Loop*.

A utilização de um ecossistema de observabilidade composto por Prometheus e Grafana foi fundamental para o diagnóstico, permitindo que a degradação do Event Loop fosse medida em tempo real. Isso sublinha que a detecção de ataques L7 não pode se basear apenas em métricas de rede, mas deve priorizar a saúde interna da aplicação.

As propostas de melhoria arquitetural — a implementação de *Load Balancers* e *Rate Limiting* via NGINX *reverse proxy* — são essenciais para transformar a arquitetura vulnerável em um sistema resiliente. A etapa crítica na implementação do *Rate Limiting* em *containers* é garantir a correta identificação do IP do cliente através do tratamento seguro do cabeçalho X-Forwarded-For. Combinando essa defesa L7 com escalabilidade horizontal e a utilização de serviços de proteção de borda baseados em nuvem, a infraestrutura atinge o nível de resiliência necessário para proteger o pilar da Disponibilidade, seguindo as melhores práticas do mercado de segurança cibernética.

7. Referências

BITTENCOURT, F.; LUCCA, G. S. MÉTODOS PARA PREVENÇÃO E DEFESA DE ATAQUES DDoS. *Revista Vincci - Periódico Científico do UniSATC*, v. 2, n. 1, 2017.

CLAVIS. A tríade da Segurança da Informação: confidencialidade, integridade e disponibilidade (CID). 2025.

AKAMAI. **O que é um ataque GET flood DDoS?**. *Akamai Glossary*. Disponível em: https://www.akamai.com/pt/glossary/what-is-a-get-flood-ddos-attack. Acesso em: 3, outubro de 2025.

AMINE, B. **Rate Limiting in a Dockerized Backend: Spring Boot and NGINX**. *Medium*, 2023. Disponível em: https://amine-benaddi.medium.com/rate-limiting-in-a-

dockerized-backend-spring-boot-and-nginx-c09fe4219743. Acesso em: 3, outubro de 2025.

AWS. **Elastic Load Balancing: Best Practices for DDoS Resiliency**. *AWS Documentation*. Disponível em: https://docs.aws.amazon.com/pt_br/whitepapers/latest/aws-best-practices-ddos-resiliency/elastic-load-balancing-bp6.html. Acesso em: 3, outubro de 2025.

AWS. **Mitigation Techniques**. *AWS Documentation*. Disponível em: https://docs.aws.amazon.com/pt_br/whitepapers/latest/aws-best-practices-ddos-resiliency/mitigation-techniques.html. Acesso em: 3, outubro de 2025.

BBHUPEN. **Monitor your Node.js App with Prometheus and Grafana using Docker**. *Medium*, 2023. Disponível em: https://bbhupen.medium.com/monitor-your-node-js-app-with-prometheus-and-grafana-using-docker-acae930c9e03. Acesso em: 3, outubro de 2025.

GRAFANA LABS. **DDoS protection: Observability, automation, and curiosity**. *Grafana Labs Blog*, 2024. Disponível em: https://grafana.com/blog/2024/09/20/ddos-protection-observability-automation-and-curiosity/. Acesso em: 3, outubro de 2025.

NGINX. **Controlling Access Proxied HTTP Traffic**. *NGINX Documentation*. Disponível em: https://docs.nginx.com/nginx/admin-guide/security-controls/controlling-access-proxied-http/. Acesso em: 3, outubro de 2025.

REDNAFI. **Rate Limiting via NGINX**. *Rednafi*. Disponível em: http://rednafi.com/go/rate_limiting_via_nginx/. Acesso em: 3, outubro de 2025.

ANDRIGHETTI, D. G. *Mitigação de ataques DDoS em redes SDN: explorando intenções como mecanismo de defesa no ONOS*. Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) – Universidade Federal de Santa Catarina, Araranguá, 2017.

POPLAVKA, S.; NIKITIN, A. Performance Comparison and Analysis of Slowloris, GoldenEye and Xerxes DDoS Attack Tools. ResearchGate, 2018.