AMEAÇAS E ESTRATÉGIAS DE DEFESA EM APIS REST: UMA ANÁLISE BASEADA NO OWASP API SECURITY

Thiago Nogueira de Oliveira, Robson de Oliveira Albuquerque
Programa de Pós-Graduação Profissional em Engenharia Elétrica – PPEE - Universidade de Brasília, Faculdade de Tecnologia, Departamento de Engenharia Elétrica, Brasília, Brasil - CEP 70910-900.
Thiago.Nogueiira@gmail.com, robson@redes.unb.br

RESUMO

Mediante os processos que levam a evolução da transformação digital e a expansão da internet voltada a serviços, as aplicações que sustentam esses serviços passaram a demandar maior integração, disponibilidade e confiabilidade. Nesse cenário, as *APIs REST* consolidaram-se como um dos principais meios de troca de dados entre sistemas distribuídos, permitindo interoperabilidade e escalabilidade. No entanto, sua ampla adoção trouxe novos riscos de segurança, especialmente devido ao tráfego de dados sensíveis e à previsibilidade de *endpoints*. O presente trabalho aborda os desafios e ameaças relacionados à segurança de *APIs REST*, com base em vulnerabilidades descritas pelo *OWASP API Security Top 10* (2023), como a *Broken Object Level Authorization* (BOLA), a autenticação quebrada e a exposição excessiva de dados. A partir de um estudo de caso sobre uma falha do tipo BOLA identificada na corretora Coinbase, demonstra-se como a ausência de validações adequadas pode comprometer a integridade de sistemas críticos e gerar impactos econômicos expressivos. Além da análise técnica, o artigo discute medidas de mitigação, destacando a importância dos três pilares da segurança de APIs (governança, monitoramento e testes), bem como o papel de ferramentas como API *Gateways* e práticas DevSecOps. Conclui-se que, diante da crescente complexidade e do aumento da superfície de ataque, a segurança de APIs deve ser tratada como elemento central no ciclo de vida das aplicações e serviços que a utilizam como base de troca de dados, exigindo controles técnicos robustos, processos de governança estruturados e estratégias contínuas de monitoramento.

Palavras-chave: APIs REST; Segurança; OWASP; Vulnerabilidades; Mitigação;

1 Introdução

A expansão dos processos que habilitam o conceito de transformação digital e a crescente dependência de sistemas baseados em aplicações conectadas ampliaram significativamente o uso de *Application Programming Interfaces* (APIs) como elemento parte da troca de dados entre diferentes plataformas e serviços. Essa interconexão, presente em sistemas e serviços que são utilizados tanto no setor público quanto no

privado, gera um ecossistema no qual dados sensíveis e operações críticas trafegam constantemente pela internet. Nesse contexto, a segurança de APIs torna-se uma preocupação relevante, visto que falhas nesse ambiente podem resultar em vazamento de informações, indisponibilidade de serviços e impactos diretos para organizações e usuários finais.

No Brasil, o movimento em direção à digitalização de serviços, aliado à legislação voltada à proteção de dados pessoais — como a Lei Geral de Proteção de Dados (LGPD) — intensifica a necessidade de práticas adequadas de segurança para APIs, estabelecendo padrões mínimos de conformidade. No entanto, a crescente complexidade desses ambientes, associada ao volume de integrações, impõe desafios na identificação, mitigação e monitoramento das vulnerabilidades existentes. As APIs REST, por serem amplamente adotadas em aplicações modernas, representam um alvo constante para agentes maliciosos.

O relatório *Open Web Application Security Project* (OWASP) API Security Top 10 (2023) evidencia que vulnerabilidades como *Broken Object Level Authorization* (BOLA), Autenticação Quebrada e Exposição Excessiva de Dados são recorrentes e frequentemente exploradas em ataques cibernéticos a sistemas e serviços digitais (vide tabela 1). Ainda assim, observa-se que muitos ambientes subutilizam mecanismos de segurança ou implementam controles de forma ineficaz, dificultando a prevenção de ameaças e sua consequente exploração em função das vulnerabilidades existentes e não tratadas corretamente, seja por processos de análise de riscos e a sua mitigação ou por incapacidade de monitoramento e testes que deveriam ser feitos de forma constante.

Tabela 1 –Top 3 Riscos de Segurança em APIs segundo o OWASP (2023)

Identificador	Vulnerabilidade	Descrição resumida
API1:2023	*Broken Object Level Authorization (BOLA)*	Falhas na autorização de nível de objeto, permitindo acesso indevido a recursos de outros usuários.
API2:2023	*Broken Authentication*	Mecanismos de autenticação fracos ou mal implementados que permitem acesso não autorizado.
API3:2023	*Broken Object Property Level Authorization (BOPLA)*	Exposição ou manipulação indevida de propriedades internas de objetos.

Portanto, a integração de boas práticas de segurança, monitoramento contínuo e automação de processos de detecção configura-se como uma abordagem essencial para reduzir riscos em APIs. Por outro lado, técnicas de auditoria, testes de segurança

automatizados e estratégias de defesa baseadas em camadas complementam e permitem, não apenas proteger os serviços, mas também garantir maior resiliência diante de ataques sofisticados.

Considerando os pontos apresentados, este artigo discute sobre desafios, ameaças, estratégias de mitigação em APIs REST e um estudo de caso com foco nas recomendações do OWASP e nas práticas adotadas no mercado. Assim, o objetivo deste artigo é examinar os principais desafios e ameaças de segurança em APIs REST, com base nas recomendações do OWASP, e consolidar diretrizes conceituais que contribuam para o fortalecimento das práticas de segurança em APIs REST no contexto organizacional.

Este artigo está estruturado conforme se segue. Após uma breve introdução, o capítulo 2 apresenta a fundamentação teórica e os trabalhos relacionados. O capítulo 3 descreve a metodologia adotada. O capítulo 4 apresenta o estudo de caso. O capítulo 5 traz as discussões. Por fim, o capítulo 6 apresenta fa o fechamento das ideias com as conclusões.

2 **Fundamentação Teórica e Revisão da Literatura**

Este capítulo apresenta os tópicos necessários ao entendimento sobre as tecnologias, descrição de termos e trabalhos relacionados a esta análise.

2.1 **Descrição sobre APIs**

As APIs consolidaram-se como componentes essenciais no desenvolvimento de sistemas distribuídos, apoiadas nas tecnologias e protocolos que estruturam a internet. Elas permitem a troca de dados e informações entre diferentes plataformas de forma padronizada, possibilitando a construção de soluções escaláveis e integradas (SHIMONI et al., 2021). Entre os modelos de troca de dados utilizados em APIs, destaca-se a arquitetura *Representational State Transfer* (REST), proposta por Fielding (2000), que adota o protocolo *Hypertext Transfer Protocol* (HTTP) para estabelecer a comunicação entre cliente e servidor. Nesse modelo, cada recurso é identificado por um *Uniform Resource Identifier* (URI) e manipulado por métodos HTTP (GET, POST, PUT, DELETE, entre outros). Um dos princípios do protocolo HTTP é o fato de ser *stateless*, ou seja, cada requisição deve incluir todas as informações necessárias para seu processamento, sem que o servidor mantenha o estado da sessão do cliente. Essas características tornam a troca de dados mais simples e previsível, além de favorecer a manipulação de recursos, a integração entre

sistemas distintos e a separação entre *front-end* e *back-end*, o que contribui para processos de manutenção e evolução mais eficientes (DIMITROV; RADOYKOV, 2021).

REST é um estilo arquitetural para sistemas distribuídos que define princípios orientados à escalabilidade, ao uso de cache e à identificação de recursos, favorecendo o desenvolvimento de aplicações modernas e robustas. Essa arquitetura promove a interoperabilidade entre diferentes plataformas, uma vez que o protocolo HTTP é amplamente suportado por diversos sistemas operacionais e linguagens de programação. Ao contrário de protocolos proprietários, o modelo REST possibilita uma comunicação padronizada e eficiente, o que justifica sua ampla adoção em arquiteturas distribuídas e em ambientes baseados em microsserviços (OWASP, 2023).

Ao contrário da interface do usuário, que opera em um ambiente controlado, a API funciona de maneira mais aberta e direta, tornando-se um alvo potencial para ataques. Enquanto a interface do usuário apresenta apenas dados e funcionalidades específicas aos usuários, a API permite que qualquer requisição seja enviada diretamente ao sistema. Essa característica possibilita que atacantes contornem a interface do usuário e interajam com a API de forma direta, manipulando parâmetros, *endpoints* e funcionalidades conforme desejarem, caso a API. Não tenha mecanismos de controle que evitem tais possibilidades. Consequentemente, APIs que não implementam controles adequados podem se tornar super permissivas, ou seja, elas acabam por oferecer acesso indevido a dados sensíveis ou funcionalidades críticas, aumentando o risco de exploração e comprometimento do sistema (OWASP, 2023).

2.2 **Principais Vulnerabilidades (OWASP 2023).**

O *OWASP API Security Top 10 (2023)* descreve as vulnerabilidades mais comuns e críticas em APIs, ressaltando que ameaças específicas diferem substancialmente das encontradas em aplicações web tradicionais. Segundo a Salt Security (2023), apenas três vulnerabilidades — BOLA, Autenticação Quebrada e Exposição Excessiva de Dados — são responsáveis por cerca de 90% dos ataques direcionados a APIs.

A vulnerabilidade BOLA classificada pelo *OWASP API Security Top 10 (2023)* como API1:2023, ocorre quando a aplicação falha em validar a propriedade de objetos ou recursos, permitindo que um usuário acesse dados pertencentes a outro. Por se tratar de uma falha lógica, muitas vezes ela não é detectada por ferramentas automatizadas

tradicionais de teste de segurança (OWASP, 2023). Já a Autenticação Quebrada, identificada como API2:2023, está relacionada a mecanismos fracos ou ausentes, resultando em acessos não autorizados. Por fim, a Exposição Excessiva de Dados, definida como API3:2023, ocorre quando a API retorna mais informações do que o necessário, ampliando a superfície de ataque (OWASP, 2023).

Essas falhas exploram lacunas na lógica da aplicação e demandam abordagens que combinem validação contextual, autorização granular e respostas padronizadas de erro HTTP para reduzir riscos (BALL, 2022).

Os códigos HTTP desempenham papel crítico na mitigação e identificação de vulnerabilidades, pois comunicam o estado da requisição e ajudam a reduzir canais laterais (*side channels*) que podem expor informações sobre a existência ou propriedade de recursos. A tabela 2 apresenta alguns exemplos essenciais.

Tabela 2 – Códigos HTTP usados em conjunto com contextos de segurança em APIs

Código	**Significado**	**Descrição e Aplicação na Segurança de APIs**
401 – Unauthorized	Não autorizado	Indica que a requisição não foi autenticada corretamente. Geralmente utilizado quando credenciais ausentes ou inválidas são fornecidas, evitando acesso não autorizado.
403 – Forbidden	Proibido	Informa que a requisição foi autenticada, mas o usuário não possui permissão para acessar o recurso solicitado. Essencial para reforçar políticas de autorização em nível de objeto.
404 – Not Found	Não encontrado	Indica que o recurso solicitado não existe ou não está acessível. Quando usado corretamente, pode impedir a enumeração de objetos sensíveis por meio da negação de informações sobre sua existência.
405 – Method Not Allowed	Método não permitido	Alerta que o método HTTP utilizado não é suportado no endpoint. Essa resposta ajuda a mitigar tentativas de exploração por uso de métodos não autorizados.
429 – Too Many Requests	Excesso de requisições	Utilizado em estratégias de *rate limiting*, ajudando a prevenir ataques de força bruta e exploração automatizada ao limitar o número de requisições em um determinado período.

A padronização dessas respostas e o tratamento uniforme de estados distintos, por exemplo, “recurso inexistente” vs. “sem permissão”, são considerados elementos centrais de uma estratégia de segurança eficaz, reduzindo a possibilidade de diferenciação de respostas por tempo ou código e, consequentemente, mitigando vetores de exploração (OWASP, 2023).

2.3 **Os Três Pilares da Segurança de APIs**

A proteção de APIs deve se apoiar em três pilares fundamentais: governança, monitoramento e testes (CITRIX, 2024; APISEC UNIVERSITY, 2024). Esses pilares atuam de forma integrada ao longo do ciclo de vida da API, desde o projeto até a produção.

A governança estabelece políticas e processos para o desenvolvimento, documentação e versionamento de APIs, bem como para a detecção e remoção de APIs “*shadow*” (não documentadas) e “*zombie*” (obsoletas), que representam riscos significativos ao ampliar a superfície de ataque (OWASP, 2023).

O monitoramento contínuo desempenha papel essencial na fase operacional, permitindo a detecção precoce de comportamentos anômalos, abuso de *endpoints* e tentativas de exploração. Soluções baseadas em análise comportamental e aprendizado de máquina podem identificar padrões como manipulação sistemática de parâmetros ou acessos fora do perfil esperado (BALL, 2022).

Os testes de segurança complementam essa abordagem ao identificar vulnerabilidades antes que atinjam ambientes produtivos. Estratégias como testes A-B/A-B-A e *fuzzing* direcionado são eficazes na detecção de falhas de autorização em nível de objeto (BOLA) e de função *Broken Function Level Authorization* (BFLA) (SHKEDY, 2021). A integração desses testes ao pipeline de *Continuous Integration/ Continuous Delivery* (CI/CD), prática que caracteriza o conceito “shift left security”, antecipa a detecção de falhas e reduz a probabilidade de exposição em produção (BALL, 2022).

2.4 ***API Gateway* e defesa em profundidade**

API *Gateway* atua como um ponto de controle centralizado, responsável pelo roteamento, agregação de requisições e aplicação de políticas de segurança. Ele pode realizar validação de identidade por meio de chaves de API, *Open Authorization* (Oauth) ou *tokens Json Web Token* (JWT) e aplicar regras básicas de autorização (F5, 2024; REDHAT, 2024). Além disso, *gateways* modernos incorporam mecanismos

como limitação de taxa (*rate limiting*), filtragem por geolocalização e detecção de anomalias, desempenhando papel relevante na mitigação de ataques de negação de serviço distribuído (DDoS) na camada de aplicação.

Contudo, soluções de perímetro, como *Web Application Firewalls* (WAF) e *gateways*, não são suficientes para impedir ataques que exploram falhas lógicas, como a BOLA. Tais ataques contornam mecanismos superficiais e exploram lacunas diretamente no *backend*, exigindo validações em nível de objeto e autorização contextualizadas na lógica da aplicação (OWASP, 2023; BALL, 2022). Assim, recomenda-se a adoção de uma estratégia de defesa em profundidade, em que *gateways* são combinados a validações no *backend* e ao uso de códigos HTTP padronizados para respostas de erro.

Além disso, soluções complementares como *Web Application* and API *Protection* (WAP) podem reforçar a análise de *payloads* e detectar comportamentos anômalos de *bots*. Mesmo com essas camadas adicionais, a validação de propriedade e a aplicação correta de respostas HTTP — como *403 Forbidden* para operações não autorizadas ou *404 Not Found* para evitar confirmação da existência de recursos — devem permanecer no *backend* como mecanismos essenciais de proteção (BALL, 2022; SHKEDY, 2021).

2.5 **Trabalhos relacionados**

Diversos artigos técnicos e relatórios destacam a crescente relevância da segurança de APIs no cenário atual e evidenciam lacunas ainda existentes na proteção desses componentes críticos. O Relatório de Segurança da Imperva (2024) aponta que, no último ano, as APIs representaram mais de 71% de todo o tráfego web, superando o tráfego tradicional e ampliando significativamente a superfície de ataque. Esse crescimento se traduz em desafios complexos para as organizações, principalmente diante do volume expressivo de requisições — uma página corporativa média pode registrar cerca de 1,5 bilhão de chamadas de API por ano.

Além do aumento quantitativo, há uma mudança qualitativa nos vetores de ataque. De acordo com o mesmo relatório, cerca de 27% dos incidentes envolvendo APIs têm origem em ataques automatizados, frequentemente direcionados à lógica de negócios. Nesses cenários, agentes automatizados — conhecidos como *bad bots* — foram responsáveis por aproximadamente 19% dos ataques, e em ambientes com alta atividade de APIs, 56% do tráfego web é proveniente de *bots*. Esses dados

evidenciam a importância de mecanismos específicos para detecção e bloqueio de tráfego automatizado malicioso como parte da estratégia de segurança (IMPERVA, 2024).

Outro ponto recorrente na literatura (OWASP, 2023; BALL, 2022; SHKEDY, 2021; IMPERVA, 2024) é a constatação de que soluções tradicionais, embora fundamentais, são insuficientes para conter a diversidade de ameaças atuais. *WAFs*, por exemplo, são eficazes na mitigação de injeções e outros ataques estruturais, mas não conseguem distinguir requisições legítimas de abusos da lógica de negócio. Da mesma forma, *API Gateways*, embora essenciais para gerenciamento de tráfego e autenticação, não oferecem profundidade para lidar com vulnerabilidades de autorização em nível de objeto, como a BOLA (OWASP, 2023; BALL, 2022).

A literatura técnica (OWASP, 2023; BALL, 2022; SHKEDY, 2021) destaca ainda a importância do tratamento correto de códigos de status HTTP como parte integrante da defesa. Shkedy (2021) demonstra que a utilização consistente de respostas como 403 (*Forbidden*) para requisições não autorizadas e 404 (*Not Found*) para ocultar a existência de recursos pode reduzir significativamente vetores de ataque baseados em enumeração e diferenciação de respostas (*response differentiation*). A implementação de 429 (*Too Many Requests*), por sua vez, auxilia na contenção de ataques automatizados, enquanto 405 (*Method Not Allowed*) previne exploração por tentativa de métodos não suportados — aspectos frequentemente negligenciados em projetos de APIs e que podem ter impacto direto na superfície de exposição.

Pereira e Lima (2021) analisam os benefícios e desafios da adoção do modelo REST, destacando sua simplicidade de integração e eficiência na utilização da infraestrutura da web. Entretanto, os autores também alertam para o risco representado pela previsibilidade dos (URI) e métodos HTTP, que pode facilitar a descoberta de *endpoints* por agentes mal-intencionados. Essa previsibilidade permite a realização de ataques de enumeração e exploração sistemática, principalmente quando não são aplicadas técnicas de validação de entrada, autenticação robusta, autorização baseada em funções e limitação de taxa de requisições (*rate limiting*) — práticas amplamente recomendadas pelo OWASP (2023) e aprofundadas em obras ~~clássicas~~ como The *Web Application Hacker's Handbook* (STALLINGS; GROSSMAN, 2021).

No mesmo sentido, Kumar e Aravinda (2024) apresentam um conjunto de práticas eficazes para fortalecimento da autenticação e autorização em APIs REST, incluindo o uso de JWT com reivindicações (*claims*) e prazos de expiração definidos, além da

aplicação de bcrypt via biblioteca passlib para proteção de senhas. Os autores enfatizam a importância da gestão segura de chaves secretas, do uso de OAuth2 para troca de credenciais, da implementação de HTTPS, da configuração de políticas *Cross-Origin Resource Sharing* (*CORS*) adequadas e da adoção de estratégias de tratamento padronizado de erros HTTP. Essas recomendações convergem com as práticas de mitigação descritas no presente trabalho, especialmente na seção 4.2, ao abordar a necessidade de validações em nível de objeto e de respostas consistentes da API.

A obra Hacking APIs (SHKEDY, 2021) também contribui para o entendimento da exploração prática de falhas em APIs modernas. O autor destaca que vulnerabilidades como BOLA e BFLA raramente são detectadas por ferramentas automatizadas de análise estática, *Static Application Security Testing* (SAST) ou dinâmica *Dynamic Application Security Testing* (DAST), pois exigem compreensão detalhada da lógica de negócio e do fluxo de autorização. Esse achado reforça a relevância de testes de segurança direcionados a APIs e de abordagens que integrem a segurança ao ciclo de vida de desenvolvimento *(shift left)*, antecipando a detecção de falhas ainda nas fases iniciais do projeto (BALL, 2022).

De forma convergente, *The Web Application Hacker's Handbook (STALLINGS; GROSSMAN, 2021)* ressalta a importância de explorar e compreender o comportamento da aplicação sob diferentes respostas HTTP, demonstrando como a manipulação de parâmetros e a análise de diferenças sutis nas respostas podem revelar falhas de autorização ou de lógica. O livro destaca ainda a necessidade de combinar técnicas automatizadas com testes manuais para identificar vulnerabilidades não triviais — abordagem que fundamenta a metodologia adotada neste estudo e detalhada no capítulo 3.

Por fim, destaca-se a convergência entre a literatura e as constatações do estudo de caso apresentado neste artigo. Tanto os relatórios da Imperva (2024) quanto as pesquisas de Kumar e Aravinda (2024) e Shkedy (2021) enfatizam a importância de monitoramento contínuo, validação contextualizada, tratamento consistente de códigos HTTP e testes específicos para falhas de autorização em nível de objeto. Tais recomendações se alinham diretamente às medidas de mitigação discutidas na seção 4.2 e ao estudo de caso apresentado no capítulo 4, demonstrando que a defesa eficaz contra ameaças em APIs exige um conjunto integrado de controles técnicos, políticas de governança e práticas contínuas de monitoramento e auditoria.

3 **Metodologia**

Este artigo adota uma abordagem exploratória e descritiva, focada na análise técnica da vulnerabilidade BOLA, em particular na falha de segurança identificada na *exchange* Coinbase em 2019 (SALT SECURITY, 2023).

A metodologia obedeceu aos seguintes passos:

I. Inicialmente, foram coletadas informações a partir de relatórios de segurança que documentaram a descoberta da vulnerabilidade durante o programa de *Bug Bounty* da corretora. Essa etapa permitiu compreender o contexto operacional da plataforma e os mecanismos de autorização implementados na época, além de fornecer evidências técnicas do comportamento inadequado da API.

II. Em seguida, procedeu-se à análise técnica do comportamento da API durante operações de negociação. Foram examinadas as requisições HTTP e os parâmetros presentes nos payloads, como asset_id, amount, price e side. A investigação concentrou-se na resposta da API a alterações maliciosas nos identificadores de ativos, evidenciando a ausência de validação de propriedade (ownership) em nível de objeto. Essa análise permitiu compreender como a falha poderia ser explorada para realizar transações não autorizadas.

III. Para fins acadêmicos, foi conduzida uma modelagem conceitual da vulnerabilidade, baseada nas evidências descritas nos relatórios de segurança e nas análises da comunidade técnica. Essa etapa consistiu na simulação controlada de requisições HTTP, reproduzindo o cenário em que o parâmetro asset_id era manipulado para representar ativos não pertencentes ao usuário autenticado. O objetivo dessa simulação não foi executar operações reais na plataforma, mas demonstrar de forma conceitual como a ausência de validação de propriedade em nível de objeto permitia a exploração da falha, conforme detalhado no estudo de caso apresentado no Capítulo 4.

IV. Posteriormente, a vulnerabilidade foi classificada com base nos padrões de segurança de APIs e nas diretrizes do *OWASP API Security Top 10 2023*, caracterizando-se como um caso típico de BOLA. Essa análise avaliou a gravidade do risco, o impacto potencial sobre a integridade financeira e operacional da plataforma e a criticidade da ausência de controles de autorização em nível de recurso.

V. Por fim, o estudo abordou possíveis medidas de mitigação, revisando boas práticas de segurança em APIs. Entre elas, destacam-se a implementação de validação de propriedade de recursos, controles rigorosos de autorização em nível de objeto, auditoria detalhada de transações e monitoramento de comportamentos anômalos. Essas recomendações foram diretamente relacionadas à vulnerabilidade identificada e reportada nos relatórios de segurança, reforçando a importância de mecanismos robustos de controle de acesso e monitoramento para plataformas de negociação de ativos digitais.

4 Estudo de Caso

4.1 Descrição do cenário sobre a Vulnerabilidade BOLA na Coinbase

Este estudo de caso analisa a vulnerabilidade BOLA descoberta na Coinbase (REF), uma das maiores corretoras de criptomoedas em atividade, responsável por intermediar a compra, venda e armazenamento de ativos digitais. O incidente é considerado um exemplo clássico de autorização quebrada em nível de objeto, classificada como *OWASP API Security Top 10 – API1*, categoria que permanece, tanto em 2019 quanto em 2023, como a ameaça número um às APIs, destacando-se por sua gravidade e pela dificuldade de detecção.

As fontes consultadas não mencionam um *Common Vulnerabilities and Exposures* (CVE) ou um *Common Weakness Enumeration* (CWE) específico para essa vulnerabilidade. Isso se deve ao fato de que falhas em APIs raramente se enquadram nas vulnerabilidades comuns, geralmente detectadas por ferramentas automatizadas de análise estática (SAST) e dinâmica (DAST) e que recebem registros formais. Em vez disso, os atacantes exploram falhas lógicas únicas, lacunas em regras de autorização e abusos em funcionalidades específicas da API. No caso da Coinbase, a falha estava associada à validação inadequada do ID do ativo, caracterizando-se como uma vulnerabilidade lógica não tipicamente contemplada por scanners tradicionais.

4.1.1 Explicação técnica da vulnerabilidade na API da coinbase

A vulnerabilidade pode ser ilustrada conceitualmente por meio de requisições. Em uma requisição legítima, o usuário, possuindo Ethereum (ETH), realizava uma operação de venda, conforme o quadro 1.

```
Json
{
```

```
  "asset_id": "ETH-USD",
  "amount": 10,
  "price": 3000,
  "side": "sell"
}
```

Quadro 1 – Operação de venda legítimo

Entretanto, ao manipular o parâmetro asset_id, um atacante poderia forjar uma requisição maliciosa, simulando a venda de Bitcoin (BTC), ativo que não possuía, conforme o quadro 2.

O comportamento esperado seria a rejeição da operação por meio de um erro de autorização ou validação de negócio, uma vez que o usuário não detinha saldo em BTC. No entanto, a API apenas verificava atributos superficiais — como quantidade e preço — sem aplicar uma validação de autorização em nível de objeto, que garantisse que o recurso (neste caso, o ativo digital Bitcoin) estivesse realmente associado à conta do usuário autenticado.

```
Json
{
  "asset_id": "BTC-USD",
  "amount": 10,
  "price": 43000,
  "side": "sell"
}
```

Quadro 2 – Operação de venda forjada

A ausência de validação de *ownership* configurou um caso clássico de BOLA, no qual identificadores de recursos podem ser manipulados nas requisições para acessar ou modificar ativos de outros usuários sem as permissões adequadas.

Esta simples análise evidencia como falhas em mecanismos de autorização podem comprometer a segurança de APIs e destaca a importância de controles rigorosos de acesso e auditoria para plataformas de negociação de ativos digitais.

O impacto potencial dessa vulnerabilidade era extremamente significativo. O pesquisador que reportou a falha descreveu-a como “potencialmente capaz de destruir o mercado” (*potentially market nuking*). Embora não haja registros de perdas financeiras concretas associadas à exploração, em virtude da rápida identificação e

correção, a possibilidade de um atacante negociar ativos inexistentes poderia resultar em transações fraudulentas em larga escala e prejuízos financeiros substanciais.

4.2 **Solução proposta**

A mitigação da vulnerabilidade BOLA exige uma abordagem estruturada, distribuída ao longo de todo o ciclo de vida da API, desde sua concepção até a operação em produção. Com base nas recomendações de OWASP (2023), Ball (2022) e Shkedy (2021), propõe-se um modelo dividido em quatro fases: projeto e modelagem, construção e *Continuous Integration* (CI), validação pré-produção e proteção e detecção em produção, complementadas por práticas de governança e gestão contínua. Essa estrutura visa reduzir a probabilidade de exploração, melhorar a detecção precoce de falhas lógicas e fortalecer a segurança em ambientes baseados em APIs REST.

4.2.1 ***Projeto e modelagem (Design-Time)***

Na fase de projeto, a segurança deve ser tratada como um requisito funcional desde o início do ciclo de desenvolvimento. Toda operação que envolva identificadores sensíveis, como asset_id, user_id ou order_id, deve implementar obrigatoriamente um mecanismo de validação de propriedade do recurso (*ownership check*). Essa validação deve ocorrer no *backend*, consultando a fonte de verdade — como bancos de dados ou *ledgers* — para confirmar a associação do recurso ao usuário autenticado antes da execução de qualquer ação (OWASP, 2023).

Além disso, recomenda-se a padronização das respostas da API. Códigos de status e mensagens devem ser consistentes tanto para situações de “recurso inexistente” quanto para “acesso não autorizado”, minimizando o risco de exploração por canais laterais (*side channels*). Respostas distintas podem fornecer a atacantes informações sobre a existência de recursos ou estrutura interna do sistema, facilitando a enumeração e o abuso da API (BALL, 2022).

Também deve ser aplicado o princípio do menor privilégio, com políticas de autorização dinâmicas que considerem o contexto da operação, a identidade do usuário e o tipo de ação executada. Essa abordagem supera limitações de políticas estáticas e contribui para mitigar ataques baseados em manipulação de parâmetros (OWASP, 2023).

Critérios de aceitação: considera-se que a fase de projeto e modelagem atende aos requisitos de segurança quando cada endpoint que referencia identificadores

sensíveis implementa regras explícitas de validação de propriedade no backend, garantindo que os recursos acessados pertençam de fato ao usuário autenticado. Além disso, deve existir documentação técnica que estabeleça o mapeamento entre as ações executadas, as verificações realizadas e as respostas retornadas pela API, assegurando um comportamento padronizado e previsível do sistema. Essas medidas visam reduzir a probabilidade de falhas de autorização e mitigar riscos associados à exploração de vulnerabilidades como a BOLA.

4.2.2 ***Construção e integração contínua (Build-Time)***

Durante o desenvolvimento e integração contínua, é necessário adotar práticas que identifiquem falhas lógicas e exposições de forma automatizada. Recomenda-se a implementação de testes automatizados A-B e A-B-A para validar a autorização em nível de objeto. Nesse modelo, cria-se um recurso com uma conta (A) e tenta-se acessá-lo ou modificá-lo com outra (B), verificando se a autorização é corretamente negada (SHKEDY, 2021).

Outra prática essencial é o uso de *fuzzing* direcionado a parâmetros, que permite identificar comportamentos inesperados ao enviar valores manipulados para campos sensíveis. Segundo Ball (2022), o *fuzzing* aumenta significativamente a cobertura de testes e é particularmente eficaz para detectar falhas de autorização não detectadas por ferramentas SAST e DAST tradicionais.

Adicionalmente, recomenda-se a inclusão de scanners de segredos e *endpoints* sensíveis no pipeline de CI/CD, capazes de identificar credenciais expostas e URIs críticas antes da implantação (BALL, 2022).

Considera-se que a fase de construção e integração contínua atende aos requisitos de segurança quando o pipeline de CI é configurado para interromper automaticamente a implantação caso qualquer teste de autorização do tipo A-B ou A-B-A seja bem-sucedido, indicando uma falha na validação de acesso entre diferentes perfis de usuários. Além disso, a entrega de código deve ser bloqueada sempre que scanners de segurança detectarem credenciais, chaves de API ou *endpoints* sensíveis expostos no repositório, garantindo que artefatos críticos não sejam incorporados ao ambiente de produção. Essas medidas asseguram que vulnerabilidades sejam identificadas e corrigidas precocemente, fortalecendo a postura de segurança antes que a aplicação seja disponibilizada.

4.2.3 ***Validação pré-produção***

A etapa de pré-produção tem como objetivo assegurar que a implementação esteja alinhada aos requisitos de segurança definidos no projeto. Devem ser realizados testes de canal lateral, verificando a uniformidade de códigos de resposta, mensagens e tempos de execução entre cenários de “recurso inexistente” e “acesso não autorizado” (BALL, 2022). Diferenças sutis nesses elementos podem indicar a presença de vulnerabilidades exploráveis.

A padronização de mensagens de erro é igualmente essencial. Conforme enfatizado por OWASP (2023), respostas genéricas reduzem a exposição de detalhes internos da aplicação e dificultam a exploração por agentes maliciosos.

Casos de teste negativos devem abranger tentativas de manipulação de identificadores e acessos não autorizados, confirmando a eficácia dos mecanismos de validação implementados (SHKEDY, 2021).

Considera-se que a fase de validação pré-produção atende aos requisitos de segurança quando as diferenças nos tempos de resposta e nos códigos HTTP retornados pela API, ao lidar com estados distintos de recursos ou condições de acesso, permanecem dentro dos limites previamente definidos, evitando a exposição de informações sobre a lógica interna do sistema. Além disso, as mensagens de erro devem ser projetadas de forma a não revelar detalhes da estrutura interna, tecnologias utilizadas ou dados sensíveis da aplicação, reduzindo o risco de exploração por meio de canais auxiliares e dificultando o mapeamento de superfícies de ataque por agentes maliciosos.

4.2.4 ***Proteção e detecção em produção***

Após a implantação, a proteção da API deve basear-se em uma defesa em camadas, combinando mecanismos de perímetro, como WAF e *gateways*, com validações de lógica no *backend*. É fundamental que a verificação de propriedade continue ocorrendo no *backend*, pois controles de perímetro isolados não são suficientes para impedir explorações baseadas em falhas lógicas (OWASP, 2023).

O uso de limitação de taxa (*rate limiting*) e análise comportamental contribui para detectar padrões de exploração, como tentativas repetidas de manipulação de identificadores. Sistemas de telemetria e correlação de eventos devem registrar decisões de autorização e emitir alertas automáticos diante de comportamentos anômalos, permitindo respostas rápidas a incidentes (BALL, 2022).

Considera-se que a fase de proteção e detecção em produção atende aos requisitos de segurança quando os painéis de monitoramento apresentam métricas detalhadas sobre negações de requisições e os motivos específicos que as originaram — como falhas de propriedade, formato inadequado ou inexistência do recurso solicitado —, permitindo uma análise precisa dos incidentes. Além disso, os planos de resposta a incidentes devem contemplar explicitamente cenários de exploração de vulnerabilidades do tipo BOLA, com procedimentos de mitigação claramente definidos, garantindo que ações corretivas possam ser executadas de forma rápida e eficaz para preservar a integridade e a disponibilidade dos serviços expostos por meio da API.

4.2.5 ***Governança e gestão do ciclo de vida***

A gestão contínua do ciclo de vida das APIs é indispensável para manter a postura de segurança ao longo do tempo. Deve-se implementar inventários automatizados de APIs para identificar e eliminar APIs "*shadow*" (não documentadas) e "*zombie*" (obsoletas), que ampliam a superfície de ataque (OWASP, 2023).

Políticas organizacionais precisam definir critérios para autenticação, autorização, versionamento e desativação segura de APIs. Versões antigas que não implementam validações adequadas devem ser removidas ou encapsuladas com camadas adicionais de proteção. Além disso, recomenda-se a revisão periódica de regras de autorização, garantindo que permaneçam alinhadas à lógica de negócio e aos requisitos de segurança (SHKEDY, 2021).

Considera-se que a fase de governança e gestão contínua atende aos requisitos de segurança quando os *scorecards* de segurança de cada API são revisados periodicamente, permitindo avaliar a conformidade com políticas organizacionais, padrões de autenticação e práticas recomendadas de proteção. Além disso, APIs obsoletas, não documentadas ou em desacordo com os controles definidos devem ser desativadas ou submetidas à aplicação de camadas adicionais de segurança, reduzindo a superfície de ataque e prevenindo a exploração de serviços não monitorados. Essa abordagem contínua assegura a manutenção de um ecossistema de APIs alinhado às melhores práticas de segurança e às exigências regulatórias.

4.2.6 ***Métricas de eficácia***

A aferição da eficácia das medidas de segurança propostas deve ser realizada por meio de indicadores quantitativos e qualitativos que permitam avaliar a maturidade do

ambiente e a eficiência dos controles implementados. Um dos principais indicadores é a cobertura mínima de 95% dos testes de autorização do tipo A-B e A-B-A em *endpoints* críticos (BALL, 2022), assegurando que os mecanismos de controle de acesso estejam sendo devidamente verificados e aplicados. A ocorrência de incidentes decorrentes de canais laterais deve ser monitorada continuamente, com a meta de alcançar zero ocorrências por milhão de requisições, refletindo a robustez das defesas contra técnicas de exploração indireta. Da mesma forma, a taxa de rejeição por falha de propriedade deve apresentar um comportamento previsível e alinhado ao esperado, demonstrando a consistência dos mecanismos de validação implementados. Outro aspecto relevante é a inexistência de segredos expostos nos pipelines de integração e entrega contínua (CI/CD), o que evidencia a adoção de boas práticas de gestão de credenciais e reforça a segurança no ciclo de desenvolvimento. Em conjunto, esses indicadores oferecem uma visão abrangente da efetividade dos controles aplicados e subsidiam decisões estratégicas para o aprimoramento contínuo da segurança das APIs.

5 **Discussões**

A partir da análise da vulnerabilidade BOLA identificada na Coinbase se mostra como falhas críticas na camada de autorização de APIs REST geram impactos de potencial desastroso para as empresas. O estudo de caso apresentado na seção 4 demonstrou como a ausência de validação do parâmetro asset_id permitia que usuários autenticados simulassem transações de ativos que não possuíam, configurando um cenário de autorização quebrada em nível de objeto. Essa constatação corrobora os riscos descritos pela literatura e pelos relatórios especializados em segurança de APIs, especialmente no *OWASP API Security Top 10 (2023)*, que posiciona o BOLA como a ameaça mais crítica e recorrente.

5.1 **Análise Técnica da Vulnerabilidade**

A avaliação do cenário da Coinbase (2019) confirmou que a API validava apenas atributos superficiais de transações — como quantidade (*amount*) e preço (*price*) —, mas não realizava a checagem de propriedade do recurso (asset_id). Na prática, enquanto uma requisição legítima era processada de acordo com o saldo do usuário, uma requisição adulterada, mediante alteração do identificador do ativo, resultava na execução de transações inválidas. Esse comportamento ilustra exatamente a

exploração de BOLA descrita no estudo de caso e evidencia o risco de ataques lógicos em APIs de alta criticidade, como corretoras de criptomoedas.

5.2 **Classificação da Vulnerabilidade**

A falha foi categorizada como OWASP API #1 (BOLA), com impacto potencial crítico. O estudo de caso demonstrou que a vulnerabilidade afetava a integridade, ao permitir transações fraudulentas; a confidencialidade, ao abrir espaço para manipulação de recursos de outros usuários; e a disponibilidade, já que ataques em massa poderiam desestabilizar a plataforma. Esses achados reforçam estatísticas da Salt Security (2023), que indicam que BOLA, juntamente com Autenticação Quebrada e Exposição Excessiva de Dados, responde por 90% dos ataques contra APIs.

5.3 **Análise do Estudo de Caso com Suporte Bibliográfico**

A análise da vulnerabilidade (BOLA) na plataforma Coinbase apresenta convergências e divergências relevantes em relação às pesquisas recentes e relatórios técnicos da área de segurança de APIs. Os resultados observados corroboram tendências apontadas por diferentes autores e estudos, ao mesmo tempo em que evidenciam lacunas que permanecem desafiadoras na proteção de APIs REST.

O relatório da Imperva (2024) destaca a predominância crescente de ataques automatizados direcionados à lógica de negócio, em especial em ambientes que registram grandes volumes de chamadas de API. Essa constatação alinha-se diretamente ao cenário analisado no estudo de caso, no qual a manipulação do parâmetro asset_id explorou uma falha lógica crítica na camada de autorização. A exploração de lacunas na lógica de negócios, segundo o relatório, é especialmente preocupante por não ser detectada por mecanismos tradicionais de segurança baseados apenas em perímetro ou assinatura.

Pesquisas de Pereira e Lima (2021) também dialogam diretamente com o caso apresentado, ao ressaltarem que a previsibilidade de URIs e métodos HTTP pode facilitar a enumeração de endpoints e a exploração de vulnerabilidades. A manipulação do campo asset_id na API da Coinbase exemplifica esse fenômeno: a estrutura previsível do endpoint e a ausência de validações robustas possibilitaram que um agente malicioso simulasse operações com ativos não pertencentes ao usuário autenticado. Essa observação reforça a necessidade de controles de

autorização contextualizados e da aplicação de políticas dinâmicas que considerem não apenas a identidade do usuário, mas também a propriedade do recurso acessado. Por sua vez, Kumar e Aravinda (2024) defendem o uso de mecanismos de autenticação baseados em tokens JWT e políticas de expiração como pilares fundamentais para a segurança de APIs. Embora essas práticas sejam eficazes na proteção contra acessos não autorizados, a vulnerabilidade estudada demonstra que elas são insuficientes quando implementadas isoladamente. A falha de autorização em nível de objeto observada na Coinbase ocorreu mesmo em um cenário autenticado, indicando que a autenticação não garante, por si só, a proteção de recursos e que a validação de propriedade deve ser tratada como camada complementar obrigatória (OWASP, 2023).

Estudos de Ball (2022) e Shkedy (2021) avançam nessa discussão ao enfatizar a importância de testes de segurança direcionados à lógica da aplicação, como os testes A-B e A-B-A, além do uso de *fuzzing* focado na manipulação de parâmetros. Essas abordagens buscam explorar cenários em que diferentes perfis de usuários interagem com o mesmo recurso, verificando se as permissões são corretamente aplicadas. A aplicação desses métodos poderia ter identificado a vulnerabilidade da Coinbase ainda nas etapas de desenvolvimento ou pré-produção, evitando sua exposição em ambiente real. A recomendação desses autores de integrar testes de autorização no pipeline de CI/CD reforça a proposta apresentada na seção 4.2 deste trabalho, que defende a incorporação precoce e contínua de validações de segurança ao longo do ciclo de vida da API.

Além disso, a literatura mais recente, como a de OWASP (2023), alerta para o papel das APIs "*shadow*" e "*zombie*" na ampliação da superfície de ataque. Esses componentes não documentados ou obsoletos frequentemente carecem de mecanismos adequados de autenticação e autorização, tornando-se vetores de exploração. A proposta apresentada neste artigo inclui a adoção de inventários automatizados e políticas de governança contínua, em conformidade com essas recomendações, reforçando a necessidade de visibilidade e controle sobre todo o ecossistema de APIs.

Outro ponto de convergência está relacionado ao uso de soluções de perímetro, como *WAFs e API Gateways*. RedHat (2024) e F5 (2024) destacam que, embora essas soluções sejam essenciais para autenticação, limitação de taxa e filtragem de tráfego, elas não são capazes de detectar falhas lógicas complexas, como a BOLA. O caso da

Coinbase ilustra esse limite: mesmo com mecanismos de perímetro implementados, a vulnerabilidade persistiu, pois dependia de validações de propriedade que não eram executadas na camada de aplicação. Essa constatação reforça a necessidade de uma defesa em profundidade, que combine mecanismos de perímetro com validações lógicas robustas no backend.

Dessa forma, o estudo de caso apresentado neste trabalho não apenas confirma observações recorrentes na literatura sobre a gravidade e a frequência das falhas de autorização em nível de objeto, mas também amplia o debate ao propor uma estrutura integrada de mitigação. Ao associar validações técnicas rigorosas, automação de testes, governança contínua e monitoramento comportamental, a proposta apresentada converge com as recomendações mais recentes (OWASP, 2023; BALL, 2022; SHKEDY, 2021) e avança ao oferecer um modelo aplicável a diferentes contextos organizacionais.

6 **Conclusões**

A intensificação da transformação digital e a crescente dependência de APIs REST como base da integração entre sistemas ampliaram significativamente a superfície de ataque dos ambientes digitais. Embora essenciais para a interoperabilidade e a escalabilidade de aplicações modernas, as APIs introduzem desafios específicos de segurança que exigem atenção contínua e estratégias de defesa adaptadas ao seu contexto. O estudo de caso apresentado, centrado na vulnerabilidade BOLA identificada na plataforma Coinbase, demonstrou como falhas lógicas em mecanismos de autorização podem comprometer a integridade de sistemas críticos e gerar impactos técnicos, financeiros e reputacionais expressivos.

A análise do caso evidenciou que vulnerabilidades como o BOLA diferem substancialmente de falhas estruturais tradicionalmente catalogadas em bases como CVE ou CWE, por explorarem lacunas de lógica de negócio e ausência de validações em nível de objeto. Essa característica reforça a necessidade de incorporar práticas de segurança contínua e de adotar a abordagem "*Shift Left*", integrando a segurança desde as fases iniciais do ciclo de desenvolvimento (SDLC) e reduzindo a probabilidade de que falhas críticas cheguem ao ambiente produtivo.

Além disso, verificou-se que mecanismos de perímetro — como *WAFs e API Gateways* —, embora relevantes para proteção inicial, não são suficientes para mitigar

ameaças que exploram deficiências internas na lógica de autorização, exigindo a implementação de controles complementares diretamente no backend da aplicação.

Outro aspecto central identificado é a importância da governança estruturada de APIs, associada a monitoramento contínuo e testes automatizados de segurança. A ausência desses elementos contribuiu para a persistência da vulnerabilidade analisada até sua descoberta em um programa de Bug Bounty, evidenciando lacunas organizacionais e técnicas no processo de desenvolvimento e operação de APIs. A criação de políticas que assegurem documentação adequada, versionamento controlado, remoção de *APIs shadow* e *APIs zombie*, bem como a aplicação consistente de códigos HTTP (*como 403 Forbidden e 404 Not Found*), são medidas essenciais para reduzir a superfície de exposição e fortalecer a resiliência do ecossistema.

A análise do caso também revelou a existência de um descompasso entre as diretrizes normativas disponíveis e a aplicação prática das recomendações em ambientes corporativos. Superar essa lacuna requer o engajamento da alta gestão, a consolidação de processos *DevSecOps*, o uso de frameworks reconhecidos, como o *OWASP API Security Top 10*, e a promoção de uma cultura organizacional voltada à segurança, sustentada por treinamentos contínuos e políticas claras de gestão do ciclo de vida de APIs.

Portanto, percebe-se que vulnerabilidades relacionadas à autorização em nível de objeto, como a BOLA, representam riscos de alto impacto e difícil detecção, podendo comprometer a disponibilidade, a confidencialidade e a integridade de sistemas críticos. O caso analisado demonstrou que a combinação de validações lógicas rigorosas, controles técnicos robustos, tratamento padronizado de erros HTTP, monitoramento em tempo real e governança eficaz constitui a estratégia mais adequada para reduzir a probabilidade e o impacto de ataques direcionados a APIs.

Como perspectiva para pesquisas futuras, recomenda-se o aprofundamento no uso de métodos de detecção de anomalias baseados em aprendizado de máquina e inteligência artificial, capazes de distinguir tráfego legítimo de ataques semânticos e comportamentais. Além disso, a criação de metodologias automatizadas de testes de segurança focadas em falhas de autorização e fluxos de negócio complexos pode elevar significativamente a eficácia das defesas. Por fim, investigações sobre mecanismos automatizados de inventário e governança de APIs apresentam potencial

para mitigar riscos associados a APIs não documentadas ou obsoletas, fortalecendo a postura de segurança das organizações no cenário digital atual.

Referências bibliográficas

APIsec UNIVERSITY. API Security Fundamentals. Disponível em: https://www.akamai.com/site/en/documents/white-paper/2024/api-security-fundamentals.pdf. Acesso em: 10 Agosto 2025.

CITRIX. API Security Best Practices. 2024. Disponível em: https://documents.trendmicro.com/images/TEx/articles/Research-Paper-API-Security_100824.pdf. Acesso em: 12 Agosto 2025.

CMC. Towards Secure APIs: A Survey on RESTful API Vulnerability Detection. Computer Modeling in Engineering & Sciences, v. 84, n. 3, 2025. Disponível em: https://www.techscience.com/cmc/v84n3/63208/html. Acesso em: 20 Agosto 2025.

DIMITROV, S.; RADOYKOV, L. RESTful API design principles. Journal of Computer Applications, v. 15, n. 4, 2021.

F5. API Security and API Gateway. 2024. Disponível em: https://www.f5.com/. Acesso em: 29 set. 2025.

FIELDING, R. T. Architectural Styles and the Design of Network-based Software Architectures. Tese (Doutorado) – University of California, Irvine, 2000.

IMPERVA. The State of API Security in 2024. 2024. Disponível em: https://www.imperva.com/resources/resource-library/reports/the-state-of-api-security-in-2024/. Acesso em: 05 set. 2025.

KUMAR, S.; ARAVINDA, R. Security measures implemented in RESTful API Development. ResearchGate, 2024. Disponível em: https://www.researchgate.net/publication/384461158_Security_measures_implemented_in_RESTful_API_Development. Acesso em: 07 set. 2025.

OWASP. API Security Project. 2023. Disponível em: https://owasp.org/www-project-api-security/. Acesso em: 05 set. 2025.

PEREIRA, L. F.; LIMA, M. A. Desafios e vulnerabilidades em APIs REST. Revista de Segurança em Redes, v. 12, n. 2, 2021.

REDHAT. Securing APIs with API Gateway. 2024. Disponível em: https://www.redhat.com/. Acesso em: 07 set. 2025.

SALT SECURITY. The 2023 State of API Security. 2023. Disponível em: https://www.traceable.ai/2023-state-of-api-security. Acesso em: 29 set. 2025.

SHIMONI, Y. et al. REST API security practices. ACM Digital Library, 2021. Disponível em: https://dl.acm.org/doi/full/10.1145/3735968. Acesso em: 07 set. 2025.

THE HACKER NEWS. APIs drive majority of Internet traffic. The Hacker News, mar. 2024. Disponível em: https://thehackernews.com/2024/03/apis-drive-majority-of-internet-traffic.html. Acesso em: 10 set. 2025.

TREND MICRO. Cyber Risk Report 2025. 2025. Disponível em: https://www.trendmicro.com/vinfo/us/security/news/threat-landscape/trend-2025-cyber-risk-report. Acesso em: 10 set. 2025.

Oliveira, Álvaro Gabriel Gomes de. Construção de aplicações distribuídas utilizando-se de APIs REST. Monografia (Bacharelado em Ciência da Computação) — Universidade do Estado do Rio Grande do Norte (UERN), Mossoró, 2018. Disponível em: https://di.uern.br/tccs/html/ltr/PDF/014006456.pdf . Acesso em: 29 set. 2025.

CMC. Towards Secure APIs: A Survey on RESTful API Vulnerability Detection. Computer Modeling in Engineering & Sciences, v. 84, n. 3, 2023. Disponível em: https://www.techscience.com/cmc/v84n3/63208/html. Acesso em: 29 set. 2025.

THE HACKER NEWS. APIs drive majority of Internet traffic. The Hacker News, mar. 2024. Disponível em: https://thehackernews.com/2024/03/apis-drive-majority-of-internet-traffic.html. Acesso em: 10 set. 2025.

BALL, J. Hacking APIs: Breaking Web Application Programming Interfaces. Hoboken: Wiley, 2022.

PORTSWIGGER; STUTTARD, D.; PINTO, M. The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws. 2. ed. Indianapolis: Wiley, 2011.

SHKEDY, E. Hacking APIs. Shelter Island: Manning Publications, 2021.