UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Mario Silva

Análise de Segurança em Aplicações Next.js: Um Estudo de Caso Baseado nas Diretrizes da OWASP Top 10

Recife

2026

Mario Silva

Análise de Segurança em Aplicações Next.js: Um Estudo de Caso Baseado nas Diretrizes da OWASP Top 10

Monografia apresentada ao Curso de Bacharelado em Ciências da Computação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciências da Computação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Computação

Curso de Bacharelado em Ciências da Computação

Orientador: Rafael Perazzo Barbosa Mota

Recife

2026

Dados Internacionais de Catalogação na Publicação
Sistema Integrado de Bibliotecas da UFRPE
Gerada automaticamente, mediante os dados fornecidos
pelo(a) autor(a)

S586a Silva, Mario Leandro Batista da.
Análise de segurança em aplicações Next.js : um estudo de caso baseado nas diretrizes da OWASP Top 10 / Mario Leandro Batista da Silva. – Recife, 2026.
38 f.; il.

Orientador(a): Rafael Perazzo Barbosa Mota.

Trabalho de Conclusão de Curso (Graduação) – Universidade Federal Rural de Pernambuco, Bacharelado em Ciência da Computação, Recife, BR-PE, 2026.

Inclui referências.

1. Segurança Web. 2. OWASP Top 10. 3. Nextjs. 4. Calcom 5. Vulnerabilidades. I. Mota, Rafael Perazzo Barbosa, orient. II. Título

CDD 004

Mario Leandro Batista da Silva

Análise de Segurança em Aplicações Next.js: Um Estudo de Caso Baseado nas Diretrizes da OWASP Top 10

Monografia apresentada ao Curso de Bacharelado em Ciências da Computação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciências da Computação

Aprovado em: 05/02/2026

BANCA EXAMINADORA

Prof. Rafael Perazzo Barbosa Mota (Orientador)
Universidade Federal Rural de Pernambuco - UFRPE

Prof. Fernado Antonio Aires Lins (Examinador Interno)
Universidade Federal Rural de Pernambuco - UFRPE

Agradecimentos

Agradeço, primeiramente, ao Prof. Rafael Perazzo Barbosa Mota pelo apoio, incentivo e paciência ao longo de toda esta jornada acadêmica. Aos meus familiares, em especial à minha mãe, à minha irmã e à minha avó, pelo constante incentivo e apoio incondicional durante toda a minha formação. Aos meus amigos, que estiveram ao meu lado e me ajudaram ao longo de toda essa trajetória, meu sincero agradecimento.

“A persistência é o caminho do êxito.”
(Charles Chaplin)

Resumo

A crescente digitalização dos serviços e a manipulação de dados sensíveis em plataformas online tornaram a segurança de aplicações Web uma prioridade crítica. Contudo, a rápida adoção de frameworks de renderização híbrida, como o Next.js, introduziu uma nova superfície de ataque ao deslocar a lógica de execução do cliente para o servidor (SSR), criando uma lacuna na literatura acadêmica que predominantemente foca em arquiteturas legadas ou Client-Side puras. Este trabalho apresenta uma análise de segurança da plataforma de agendamento *open-source* Cal.com, desenvolvida com o *framework* moderno Next.js. O objetivo principal foi identificar vulnerabilidades de segurança utilizando a metodologia de testes de caixa-preta e a ferramenta automatizada OWASP ZAP (*Zed Attack Proxy*), tendo como referencial as categorias de risco estabelecidas no OWASP Top 10:2025 Release Candidate. Os testes foram conduzidos em um ambiente controlado utilizando contêineres Docker. A varredura revelou um total de 277 alertas de segurança, incluindo uma vulnerabilidade de risco alto (Include Server Side - ISS) e quinze de risco médio, relacionadas principalmente a falhas de configuração de segurança e proteção insuficiente de dados. A análise demonstrou que, apesar das proteções nativas oferecidas pelo Next.js, como Server-Side Rendering (SSR) e Middleware, a implementação padrão e a integração com APIs externas ainda podem expor a aplicação a riscos significativos se não forem devidamente configuradas. Este estudo contribui para a área de segurança de software ao validar a aplicabilidade das novas diretrizes da OWASP em arquiteturas modernas e fornecer recomendações práticas de mitigação, como a implementação de Content Security Policy (CSP) e a sanitização de entradas, servindo como referência para desenvolvedores e auditores de segurança.

Palavras-chave: Segurança Web, OWASP Top 10, Next.js, Cal.com, Vulnerabilidades, OWASP ZAP.

Abstract

The increasing digitalization of services and the handling of sensitive data on online platforms have made Web application security a critical priority. However, the rapid adoption of hybrid rendering frameworks, such as Next.js, has introduced a new attack surface by shifting execution logic from the client to the server (SSR). This has created a gap in academic literature, which predominantly focuses on legacy or pure Client-Side architectures. This study presents a security analysis of the open-source scheduling platform Cal.com, developed with the modern Next.js framework. The primary objective was to identify security vulnerabilities using black-box testing methodology and the OWASP ZAP (Zed Attack Proxy) automated tool, using the risk categories established in the OWASP Top 10:2025 Release Candidate as a benchmark. The tests were conducted in a controlled environment using Docker containers. The scan revealed a total of 277 security alerts, including one high-risk vulnerability (Include Server Side - ISS) and fifteen medium-risk vulnerabilities, primarily related to security misconfigurations and insufficient data protection. The analysis demonstrated that, despite the native protections offered by Next.js, such as Server-Side Rendering (SSR) and Middleware, default implementations and integration with external APIs can still expose the application to significant risks if not properly configured. This study contributes to the field of software security by validating the applicability of the new OWASP guidelines in modern architectures and providing practical mitigation recommendations, such as the implementation of Content Security Policy (CSP) and input sanitization, serving as a reference for developers and security auditors.

Keywords: Web Security, OWASP Top 10, Next.js, Cal.com, Vulnerabilities, OWASP ZAP.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
CDNs	Content Delivery Networks
CI/CD	Continuous Integration/Continuous Delivery
DAST	Dynamic Application Security Testing
DBMS	Database Management System
DDoS	Distributed Denial of Service
DoS	Denial of Service
ECB	Electronic Code Book
HTTP	Hypertext Transfer Protocol
IDOR	Insecure Direct Object Reference
IVs	Initialization Vectors
JWT	JSON Web Token
MFA	Multi-Factor Authentication
OS	Operating System
OWASP	Open Web Application Security Project
PHI	Protected Health Information
PII	Personally Identifiable Information
SAST	Static Application Security Testing
SDL	Secure Development Lifecycle
SEO	Search Engine Optimization
SOC	Security Operations Center
SQL	Structured Query Language
SSG	Static Site Generation
SSL	Secure Sockets Layer

SSO	Single Sign-On
SSR	Server-Side Rendering
TLS	Transport Layer Security
URL	Uniform Resource Locator
ZAP	Zed Attack Proxy

Sumário

1 Introdução

Com a popularização das aplicações Web em diferentes setores como comércio eletrônico, educação a distância e gestão de serviços públicos, a segurança da informação tornou-se uma preocupação fundamental para garantir a proteção de dados sensíveis e a confiabilidade dos sistemas (LALA; KUMAR; T., 2021). Vulnerabilidades em aplicações Web podem permitir que agentes maliciosos explorem falhas de autenticação, manipulem dados e comprometam a integridade de serviços online, causando prejuízos financeiros, legais e reputacionais para organizações e usuários (NURBOJATMIKO et al., 2022).

Dentre as iniciativas mais relevantes para o fortalecimento da segurança de aplicações Web, destaca-se a *Open Web Application Security Project* (OWASP) (OWASP, 2025), que desde 2003 mantém o documento OWASP Top 10, uma lista que reúne as vulnerabilidades mais críticas em aplicações Web, atualizada periodicamente para refletir os riscos mais atuais. O OWASP Top 10 tornou-se uma referência mundial para profissionais de segurança, desenvolvedores e equipes de gestão de TI, sendo utilizado como base para auditorias, treinamentos e desenvolvimento seguro de software (QUINCOZES et al., 2024).

Nos últimos anos, o ecossistema de desenvolvimento Web tem sido amplamente dominado pelo React, uma biblioteca JavaScript criada pela Meta (Facebook) e voltada para a construção de interfaces de usuário baseadas em componentes reutilizáveis. O React trás conceitos como *Virtual DOM* e renderização declarativa, permitindo a criação de aplicações responsivas e escaláveis (Meta Platforms, Inc., 2025). Entretanto, sua ampla adoção também expôs novos desafios de segurança, como injeção de código malicioso em componentes, armazenamento inseguro de dados no cliente e falta de controle de sessão quando integrado a *back-ends* externos (SRIVASTAVA et al., 2024).

Com o intuito de superar algumas limitações do React, surgiu o Next.js, um *framework* que estende suas funcionalidades e oferece uma abordagem *full stack* ao incorporar *Server-side rendering* (SSR), *Static Site Generation* (SSG) e rotas baseadas em arquivos (Vercel Inc., 2025). Essas inovações aprimoram o desempenho e o *Search Engine Optimization* (SEO), mas também introduzem novos vetores de vulnerabilidade, pois parte do código é executada no servidor, ampliando a superfície de ataque e exigindo práticas mais rigorosas de segurança (SRIVASTAVA et al., 2024).

De acordo com Srivastava et al. (SRIVASTAVA et al., 2024), o Next.js representa uma evolução significativa dentro do ecossistema React, unificando o desenvolvimento *front-end* e *back-end* e simplificando o fluxo de desenvolvimento. No entanto, suas funcionalidades como *middleware*, *Application Programming Interface* (API) *Routes* e *data fetching*, quando configuradas incorretamente, podem expor dados sensíveis ou permitir ataques de injeção. Por isso,

compreender como as diretrizes da OWASP se aplicam nesse contexto é essencial para o desenvolvimento de aplicações seguras.

Pesquisas recentes reforçam esse ponto. Muhammad et al. (MUHAMMAD; SYAHPUTRA; ARDIMANSYAH, 2025) demonstraram que a adoção de práticas seguras e bibliotecas complementares como *Yup Validation, Nookies, bcrypt e Middleware*, pode reduzir significativamente a exposição a vulnerabilidades críticas. No estudo, o número de falhas categorizadas pelo OWASP Top 10 foi reduzido de 12 para 7, e o risco médio de exploração caiu de 7,9 para 5,57, evidenciando que, embora o Next.js possua mecanismos avançados, sua segurança depende fortemente das escolhas e práticas do desenvolvedor.

Apesar da importância do OWASP Top 10, ainda existem lacunas significativas na forma como suas recomendações são implementadas na prática. Muitos estudos concentram-se em análises pontuais de sistemas governamentais, empresariais ou acadêmicos, frequentemente baseados em versões anteriores da lista, deixando de abordar a aplicação prática das recomendações mais recentes no desenvolvimento de aplicações Web, como as do Next.js (QUINCOZES et al., 2024; CHOIRIYAH; QOMARIASIH, 2023; NURBOJATMIKO et al., 2022). Estudos de caso realizados com base em listas anteriores ainda não contemplam plenamente as atualizações mais recentes.

Dessa forma, esta pesquisa propõe-se a examinar o nível de vulnerabilidade e os principais problemas de segurança na plataforma de agendamento open-source Cal.com, utilizada como estudo de caso de uma aplicação Web desenvolvida com o framework Next.js, utilizando as diretrizes recomendadas pela OWASP. O objetivo é identificar vulnerabilidades recorrentes, compreender suas causas e fornecer recomendações práticas que possam auxiliar desenvolvedores e organizações na criação de aplicações mais seguras.

1.1 Justificativa

A segurança de aplicações Web é uma preocupação crescente em um cenário onde dados sensíveis são constantemente manipulados e armazenados online. De acordo com o relatório *KSB Statistics of the Year* (KASPERSKY, 2024), durante o período de novembro de 2023 até outubro de 2024, mais de 302 milhões de ataques de *malware* foram bloqueados em todo o mundo, incluindo 85 milhões de *Uniform Resource Locator* (URLs) maliciosos únicos e mais de 72 milhões de objetos maliciosos detectados pelo Anti-Virus, evidenciando o aumento expressivo de ameaças digitais no cenário global. A falta de medidas preventivas adequadas pode acarretar sérios prejuízos financeiros, legais e também reputacionais para organizações e usuários.

A realização de uma análise de conformidade com base nas diretrizes atualizadas da OWASP, uma das principais referências globais no fornecimento de práticas seguras para o desenvolvimento Web, permite identificar possíveis lacunas de segurança que podem comprometer a integridade de um sistema. De acordo com (LALA; KUMAR; T., 2021), a aplicação

prática das diretrizes da OWASP em aplicações Web permite reduzir vulnerabilidades como injeção de *scripts*, exposição de dados sensíveis e autenticação falha, evidenciando a importância da adoção de medidas preventivas no ciclo de desenvolvimento.

Existem diversos estudos na literatura que abordam a segurança em aplicações Web de maneira ampla ou com foco em diretrizes como o OWASP Top 10. Entretanto, a grande maioria desses estudos concentra-se em avaliar como determinados sistemas governamentais, empresariais ou acadêmicos se comportam quando submetidos a algumas das vulnerabilidades listadas no OWASP Top 10 2021. (QUINCOZES et al., 2024) analisou a plataforma *Moodle* por meio da ferramenta OWASP ZAP, identificando e categorizando as principais vulnerabilidades segundo o OWASP Top 10 2021, discutindo seus impactos e apresentando contramedidas para mitigá-las. Além disso, pesquisas anteriores (NURBOJATMIKO et al., 2022) analisaram vulnerabilidades em plataformas de *crowdfunding sharia* usando o OWASP ZAP, reforçando a importância de avaliar aplicações Web em diferentes contextos. Na mesma linha, outros autores (CHOIRIYAH; QOMARIASIH, 2023) aplicaram o OWASP Top 10 2021 para investigar vulnerabilidades em sites de serviços de saúde na Indonésia, apontando problemas como *Broken Access Control* e *Security Misconfiguration*, e propondo soluções para mitigá-los. Contudo, há escassez de estudos que explorem o OWASP Top 10 em aplicações construídas especificamente com o *framework* Next.js, cuja arquitetura híbrida traz desafios particulares de segurança. Por meio de um estudo de caso, atualizado conforme a lista TOP 10 2025 Release Candidate (OWASP, 2025), torna-se possível aplicar essas diretrizes na prática, avaliar o grau de maturidade da aplicação analisada e propor melhorias concretas de acordo com os problemas de segurança mais atuais.

Dessa forma, este trabalho justifica-se pela necessidade de avaliar a segurança de aplicações desenvolvidas com Next.js fundamentada nas diretrizes mais recentes da OWASP Top 10 Release Candidate, para o ano de 2025, contribuindo para preencher essa lacuna na literatura. Além disso, este trabalho contribui para o fortalecimento da segurança em ambientes Web e promove a adoção consciente de boas práticas no desenvolvimento de software. A pesquisa busca analisar a manifestação das principais vulnerabilidades descritas pela OWASP, propor estratégias de mitigação aplicáveis ao ambiente Next.js e fomentar a adoção de práticas de desenvolvimento seguro. Assim, espera-se fornecer um material de referência que auxilie desenvolvedores, pesquisadores e profissionais de segurança a compreender melhor as fragilidades desse ecossistema e aprimorar a proteção de aplicações Web modernas.

1.2 Problema de Pesquisa

Considerando a crescente adoção do *framework* Next.js no desenvolvimento de aplicações Web modernas e o aumento das ameaças cibernéticas registradas nos últimos anos, torna-se necessário compreender como as vulnerabilidades descritas pela OWASP Top 10 se manifestam nesse ambiente específico. Assim, o problema que norteia esta pesquisa pode ser formulado da

seguinte maneira:

Quais as vulnerabilidades mais críticas listadas pela OWASP Top 10 se manifestam em aplicações desenvolvidas com o *framework* Next.js, e quais estratégias podem ser aplicadas para sua mitigação de forma eficaz?

1.3 Objetivos

Este trabalho tem como objetivo realizar uma análise de segurança, tendo como objeto de estudo uma aplicação Web desenvolvida com o *framework* Next.js, com base na versão mais recente do OWASP Top 10, identificando vulnerabilidades críticas, avaliando seus riscos e propondo contramedidas eficazes para mitigar possíveis ameaças cibernéticas. A pesquisa busca oferecer uma contribuição prática para o aprimoramento da segurança em aplicações desenvolvidas com tecnologias modernas, promovendo a adoção de boas práticas recomendadas pela OWASP. Os objetivos específicos deste trabalho são pontuadas a seguir:

- Identificar e classificar as vulnerabilidades presentes na aplicação desenvolvida em Next.js, considerando o contexto de sua arquitetura híbrida e os recursos oferecidos pelo *framework*;
- Analisar os riscos de exploração associados às vulnerabilidades detectadas, com base nas categorias do OWASP Top 10 (versão atualizada);
- Apresentar as causas técnicas de cada vulnerabilidade encontrada, discutindo seus impactos potenciais sobre a integridade, confidencialidade e disponibilidade do sistema;
- Propor medidas de mitigação adequadas para reduzir ou eliminar as vulnerabilidades identificadas, com foco nas particularidades do ambiente Next.js;
- Oferecer orientações práticas para desenvolvedores e administradores, visando o fortalecimento da segurança de aplicações que utilizam o *framework* Next.js;
- Comparar o nível de segurança da aplicação antes e depois da aplicação das medidas corretivas, demonstrando a eficácia das recomendações propostas.

Com esses objetivos, pretende-se fornecer informações relevantes e práticas para a segurança das aplicações Web, contribuindo para o fortalecimento das políticas e processos de proteção adotados por desenvolvedores e administradores. Dessa forma, busca-se não apenas garantir a integridade, confidencialidade e disponibilidade dos sistemas, mas também promover uma cultura de desenvolvimento seguro, alinhada às melhores práticas recomendadas pela OWASP e às necessidades atuais de proteção de dados e continuidade dos serviços digitais.

O restante deste trabalho está organizado da seguinte maneira: no Capítulo 2, apresenta-se o referencial teórico necessário para o entendimento da pesquisa. No Capítulo 3, são detalhados os procedimentos metodológicos adotados. O Capítulo 4 é dedicado à análise prática, que ocorre em duas etapas: na primeira, são identificadas e classificadas as vulnerabilidades presentes na aplicação com base nas categorias da OWASP Top 10; na segunda, são apresentadas as contramedidas implementadas e a comparação entre o estado inicial (*vulnerable*) e o ambiente corrigido (*hardened*), avaliando a eficácia das medidas aplicadas. Por fim, no Capítulo 5, são apresentadas as conclusões obtidas a partir dos resultados.

2 Referencial Teórico

Neste capítulo serão apresentados os principais conceitos fundamentais, os trabalhos correlatos e as tecnologias aplicadas durante este trabalho.

2.1 Conceitos Fundamentais

A segurança de *websites* é uma preocupação crítica para organizações e desenvolvedores, pois vulnerabilidades não corrigidas se tornam alvos fáceis para *hackers* (LALA; KUMAR; T., 2021). A exposição a essas vulnerabilidades pode resultar em perdas financeiras ou comprometer dados sensíveis (LALA; KUMAR; T., 2021).

No contexto da segurança da informação, três propriedades fundamentais são recorrentemente abordadas na literatura:

- **Confidencialidade**: Proteção contra acesso não autorizado.
- **Integridade**: Garantia de que os dados não foram alterados.
- **Disponibilidade**: Assegurar que os sistemas permaneçam acessíveis aos usuários.

Essas propriedades são amplamente reforçadas no estudo (QUINCOZES et al., 2024), que as apresenta como pilares essenciais no desenvolvimento seguro de software e tríade principal da segurança da informação.

2.1.1 Vulnerabilidades e Riscos

Vulnerabilidades são falhas de segurança que, quando exploradas, podem levar a sérios comprometimentos, e aquelas que chegam a ser implantadas em produção representam uma ameaça contínua (QUINCOZES et al., 2024). É fundamental entender que a vulnerabilidade é uma condição preexistente, ela não é o ataque em si, mas sim a causa que o possibilita, como um software desatualizado ou uma configuração de senha fraca. Para que essa falha se manifeste como um risco real, ela deve ser explorada por uma ameaça e um ator de ameaça. Dentro do escopo deste trabalho, buscamos estudar as vulnerabilidades do OWASP Top 10 na plataforma Cal.com

2.1.2 Ferramentas de Scan

Para a detecção e análise sistemática de vulnerabilidades, são empregadas ferramentas de scan automatizado (QUINCOZES et al., 2024). A ferramenta de eleição para este estudo é

o (*OWASP ZAP Zed Attack Proxy*), uma solução de código aberto amplamente utilizada para testes de segurança em aplicações web (QUINCOZES et al., 2024; MUHAMMAD; SYAHPUTRA; ARDIMANSYAH, 2025; CHOIRIYAH; QOMARIASIH, 2023). Como um dos projetos carro-chefe da OWASP, ele opera como um ”homem no meio”(*proxy*) que intercepta e inspeciona o tráfego entre o navegador e a aplicação web. Além de atuar como um *proxy* manual para inspeção detalhada, o ZAP oferece recursos automatizados, como o *Scanner* Ativo e o *Scanner* Passivo, que buscam sistematicamente falhas de segurança comuns. Sua natureza *open source* e sua vasta comunidade de desenvolvedores e *testers* garantem que a ferramenta seja constantemente atualizada com as mais recentes técnicas de ataque e detecção de vulnerabilidades.

2.1.3 Ataques

Ataques são as ações exploratórias que buscam tirar proveito das vulnerabilidades de segurança, e suas consequências podem ser severas, incluindo perdas financeiras (QUINCOZES et al., 2024). Essas ações podem se manifestar de diversas formas, desde técnicas de engenharia social, como o *phishing*, que manipula a confiança humana para obter informações confidenciais até explorações altamente sofisticadas, como a Injeção de SQL ou o uso de *Malware*. O *malware* inclui programas como o *Ransomware*, que criptografa dados e exige um resgate para o acesso, além de Vírus, *Worms* e *Spyware*. Ataques de Negação de Serviço (DoS/DDoS) também são comuns, pois sobrecarregam servidores e tornam sistemas indisponíveis para usuários legítimos.

2.2 OWASP Top 10

A fundamentação teórica e a categorização das vulnerabilidades abordadas neste estudo baseiam-se estritamente nas diretrizes estabelecidas pela iniciativa *(OWASP Open Web Application Security Project)*, uma estrutura de código aberto dedicada a aprimorar a segurança de aplicações. A lista OWASP Top 10 é a principal iniciativa global para a conscientização sobre a segurança de aplicações web, servindo como padrão de referência para classificar e conscientizar sobre as vulnerabilidades mais críticas (OWASP, 2025; QUINCOZES et al., 2024; NURBOJATMIKO et al., 2022). A edição de referência utilizada no estudo é a OWASP Top 10:2025 Release Candidate. As seções subsequentes detalham essas categorias de risco, explorando seus mecanismos de funcionamento, impactos potenciais e as falhas de implementação que as originam, conforme preconizado pela organização.

2.2.1 Broken Access Control

Esta é uma vulnerabilidade de segurança crítica onde as políticas de acesso não são aplicadas corretamente, permitindo que usuários ajam fora de suas permissões pretendidas. Esta falha ocorre quando a aplicação não consegue validar adequadamente se um usuário possui a autoridade necessária para executar uma ação, acessar um recurso ou manipular dados (OWASP,

2025). As consequências de um controle de acesso quebrado são severas e incluem divulgação, modificação ou destruição não autorizada de dados, bem como a execução de funções restritas a usuários privilegiados (OWASP, 2025). A vulnerabilidade se manifesta de várias maneiras, sendo a Violação do Princípio do Mínimo Privilégio um erro fundamental, onde o acesso é concedido por padrão e não restrito estritamente às necessidades do usuário. As técnicas de ataque mais comuns exploram esse princípio, como o *Bypassing* através da modificação direta de identificadores na *Uniform Resource Locator* (URL), que inclui o *Insecure Direct Object Reference* (IDOR), permitindo que um usuário acesse ou manipule a conta de terceiros simplesmente alterando parâmetros (OWASP, 2025). Outras manifestações críticas envolvem a Elevação de Privilégio, onde um usuário de baixo nível consegue assumir funções de administrador, e a Manipulação de Metadados, visando obter privilégios indevidos. A correção do *Broken Access Control* exige que a aplicação adote uma abordagem de ”negar por padrão”, garantindo que todas as requisições sejam validadas no lado do servidor com base na função e nas permissões explícitas do usuário autenticado.

2.2.2 Security Misconfiguration

A vulnerabilidade de Má Configuração de Segurança, do inglês, *Security Misconfiguration* ocorre quando um sistema, aplicação, ou serviço em nuvem é configurado incorretamente do ponto de vista da segurança (OWASP, 2025). Essa falha está intimamente ligada à ausência de um processo rigoroso e centralizado para o fortalecimento da segurança (*security hardening*) em toda a aplicação, o que eleva significativamente o risco de ataque. A aplicação pode se tornar vulnerável por diversos motivos, como a ausência do fortalecimento adequado em qualquer módulo ou a configuração imprópria de permissões em serviços de nuvem (OWASP, 2025). Outras manifestações incluem a presença de recursos, portas, serviços ou contas de teste desnecessários que permanecem habilitados, ou o fato de contas padrão e suas senhas originais não terem sido alteradas. A má configuração também se revela na falta de tratamento centralizado de erros, o que resulta na exposição de mensagens de erro excessivamente informativas ou *stack traces* diretamente aos usuários. Além disso, a falha em configurar os recursos de segurança mais recentes após uma atualização, ou a priorização excessiva da compatibilidade com versões anteriores em detrimento de configurações seguras, são causas comuns. Finalmente, o uso de configurações inseguras em *frameworks* de aplicação, servidores, bibliotecas e bancos de dados, e a omissão de cabeçalhos e diretivas de segurança por parte do servidor, são indicadores claros de uma má configuração de segurança. A mitigação exige uma abordagem proativa para garantir que todos os componentes estejam definidos para o maior nível de proteção possível.

2.2.3 Software Supply Chain Failures

São quebras ou comprometimentos no processo de construção, distribuição ou atualização de software. Elas são frequentemente causadas por vulnerabilidades ou alterações maliciosas

introduzidas em códigos de terceiros, ferramentas ou outras dependências que o sistema utiliza (OWASP, 2025). Uma organização está particularmente vulnerável a essas falhas se não rastrear cuidadosamente as versões de todos os componentes utilizados, tanto *client-side* quanto *server-side*, incluindo as dependências transitivas. O risco aumenta se o software, o OS, o servidor web, o DBMS, APIs, ambientes de *runtime* ou bibliotecas estiverem vulneráveis, sem suporte ou desatualizados (OWASP, 2025). A falta de escaneamento regular de vulnerabilidades e de assinaturas de boletins de segurança relacionados aos componentes em uso é um fator de risco significativo. A vulnerabilidade é agravada pela ausência de um processo de gerenciamento de mudanças para rastrear alterações em toda a cadeia de suprimentos, englobando IDEs, extensões, repositórios de código e a forma como os artefatos são criados e armazenados. É crucial que cada parte da cadeia seja documentada e devidamente fortalecida. A falta de separação de deveres nos sistemas de *supply chain*, onde uma única pessoa pode promover código até a produção sem supervisão, e a permissão para que desenvolvedores baixem componentes de fontes não confiáveis são falhas graves.

2.2.4 Cryptographic Failures

As Falhas Criptográficas são vulnerabilidades que surgem da implementação incorreta ou insuficiente de criptografia destinada a proteger a confidencialidade e a integridade dos dados (OWASP, 2025). É uma exigência básica que os dados em trânsito sejam criptografados na camada de transporte (ex: TLS/SSL), sendo que dados sensíveis demandam criptografia adicional em repouso e na camada de aplicação. Uma aplicação é considerada vulnerável se utilizar algoritmos ou protocolos criptográficos fracos ou obsoletos como *MD5* ou *SHA1*, ou se houver falhas no gerenciamento de chaves, o que inclui o uso de chaves padrão, fracas, reutilizadas ou armazenadas diretamente em repositórios de código-fonte (OWASP, 2025). Além disso, a aplicação é insegura se a criptografia não for imposta, o que se manifesta na falta de cabeçalhos de segurança *HTTP*, se houver falhas técnicas na implementação, como o uso inseguro de vetores de inicialização (IVs), modos de operação inseguros (como ECB), ou falha na validação de certificados. Por fim, a falha em usar funções *hash* criptográficas quando necessário ou a utilização de aleatoriedade que não atende aos requisitos criptográficos comprometem ainda mais a segurança. A correção dessas falhas exige um rigoroso design criptográfico, garantindo a força dos algoritmos, o gerenciamento seguro das chaves e a validação completa de todos os componentes criptográficos.

2.2.5 Injection

A vulnerabilidade de Injeção é uma falha crítica que permite a um atacante introduzir código ou comandos maliciosos, como SQL ou comandos de shell nos campos de entrada da aplicação, enganando o sistema para que ele execute o código como se fosse legítimo, o que pode ter consequências severas (OWASP, 2025). Ocorre principalmente quando os dados fornecidos

pelo usuário não são validados, filtrados ou sanitizados, ou quando consultas dinâmicas são utilizadas diretamente no interpretador sem o tratamento adequado (OWASP, 2025). Tipos comuns incluem *SQL Injection*, *NoSQL Injection* e *OS Command Injection*. A mitigação eficaz depende da parametrização de consultas e do uso de ferramentas de segurança (SAST/DAST) no pipeline de desenvolvimento para identificar inputs vulneráveis. O fluxo desse tipo de vulnerabilidade, especificamente no caso de Injeção de SQL, pode ser visualizado na Figura 1.

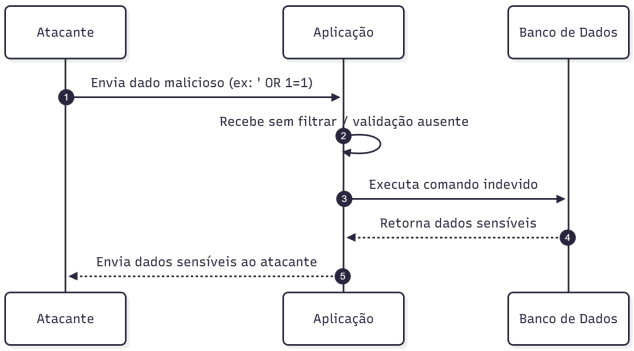

Figura 1 – Fluxo simplificado de um ataque de Injeção de SQL.

2.2.6 Insecure Design

Design Inseguro é uma categoria ampla que representa diversas fraquezas caracterizadas pela ausência ou ineficácia do design de controles de segurança (OWASP, 2025). É crucial distinguir entre design inseguro e implementação insegura, pois falhas de design possuem causas raiz, momentos de ocorrência e remediações diferentes dos defeitos de implementação. Um design seguro ainda pode conter defeitos de implementação que levam a vulnerabilidades, mas um design inseguro não pode ser corrigido nem mesmo por uma implementação perfeita, visto que os controles de segurança necessários para a defesa contra ataques específicos nunca foram criados (OWASP, 2025). Um dos fatores que contribui significativamente para essa falha é a falta de perfilhamento do risco de negócio inerente ao sistema, o que resulta na incapacidade de determinar o nível de segurança exigido. Para garantir um design seguro, é fundamental atuar em três frentes principais: a reunião de requisitos e gerenciamento de recursos, a criação ativa de um design seguro e a manutenção de um Ciclo de Vida de Desenvolvimento Seguro (SDL).

2.2.7 Authentication Failures

Elas estão presentes quando um sistema é ludibriado para reconhecer um usuário inválido ou incorreto como legítimo, indicando fraquezas nos mecanismos de verificação de identidade (OWASP, 2025). Isso ocorre frequentemente se a aplicação permite ataques automatizados como o *credential stuffing*, uso de listas de credenciais vazadas ou *password spraying*, testes com variações de senhas comuns, além de não bloquear rapidamente tentativas de força bruta

(OWASP, 2025). A vulnerabilidade também é caracterizada pela permissão de senhas fracas, padrão ou já comprometidas, bem como pelo uso de processos de recuperação de conta inseguros, como perguntas baseadas em conhecimento *"knowledge-based answers"*. O armazenamento inadequado de senhas e a ausência ou ineficácia de autenticação multifator (MFA) agravam significativamente o risco. Por fim, falhas na gestão de sessão, como a exposição de identificadores na URL, a reutilização de IDs após o login e a não invalidação correta de sessões ou *tokens* após o logout ou inatividade, completam o cenário de insegurança

2.2.8 Software or Data Integrity Failures

As Falhas de Integridade de Software e Dados ocorrem quando o código e a infraestrutura falham em proteger o sistema contra o uso de código ou dados não confiáveis como se fossem legítimos (OWASP, 2025). Esse problema se manifesta quando aplicações dependem de *plugins*, bibliotecas ou módulos provenientes de fontes, repositórios e CDNs não confiáveis (OWASP, 2025). Um vetor crítico é a presença de um *pipeline* de CI/CD inseguro que, ao negligenciar verificações de integridade, como a checagem de assinaturas digitais ou puxar artefatos de locais não verificados, abre brechas para acesso não autorizado e injeção de código malicioso (OWASP, 2025). Além disso, mecanismos de atualização automática sem verificação robusta permitem que atacantes distribuam versões comprometidas para todas as instalações. A categoria também abrange a desserialização insegura, onde objetos ou dados serializados podem ser visualizados e modificados por atacantes antes de serem processados pelo sistema.

2.2.9 Logging and Alerting Failures

A ausência de registros (*logging*) e monitoramento adequados impede a detecção de ataques e violações, tornando a resposta a incidentes lenta e ineficaz. Essa vulnerabilidade se manifesta quando eventos auditáveis críticos, como logins, tanto sucessos quanto falhas e transações de alto valor, não são registrados ou o são de forma inconsistente, e quando mensagens de erro e avisos são inadequados ou pouco claros. A integridade dos *logs* é frequentemente negligenciada, permitindo adulterações, e o armazenamento puramente local sem backups adequados compromete a análise forense. Além disso, a falha em monitorar atividades suspeitas em aplicações e APIs, somada à inexistência de processos eficazes de escalonamento e limites de alerta, deixa o sistema cego, a ponto de ferramentas de teste (DAST) não dispararem avisos e ataques ativos não serem detectados em tempo real. Riscos adicionais incluem o vazamento de informações sensíveis (PII ou PHI) através dos próprios registros, a vulnerabilidade a injeções nos sistemas de *log* por falta de codificação correta dos dados e a sobrecarga operacional das equipes de segurança (SOC) causada por um excesso de falsos positivos ou pela falta de *playbooks* e casos de uso atualizados para processar corretamente os incidentes.

2.2.10 Mishandling of Exceptional Conditions

O Manuseio Incorreto de Condições Excepcionais ocorre quando o software falha em prevenir, detectar ou responder adequadamente a situações inusitadas e imprevisíveis, resultando em travamentos, comportamentos inesperados e sérias vulnerabilidades. Esse problema surge frequentemente devido a validações de entrada deficientes, tratamento de erros tardio ou incompleto, e instabilidades ambientais, o que leva o sistema a cair em um estado desconhecido e inseguro. Essa gestão inadequada de exceções cria vetores para diversos riscos de segurança, incluindo *bugs* de lógica, *overflows*, condições de corrida e falhas de autenticação, que comprometem a confidencialidade, integridade e disponibilidade dos dados, permitindo que atacantes manipulem o tratamento de erros falho da aplicação para obter vantagens indevidas.

2.3 Next.js

A aplicação em estudo utiliza o *framework* Next.js, uma arquitetura moderna para o *front-end* web baseada em React.js (Vercel Inc., 2025). Estudos demonstram que a correta integração de seus recursos minimiza vulnerabilidades e aprimora a segurança geral (MUHAMMAD; SYAHPUTRA; ARDIMANSYAH, 2025). Dentre os principais recursos de segurança que o *framework* oferece, destacam-se o *Server-Side Rendering* (SSR), uma técnica de renderização que limita a lógica de negócio exposta ao lado do cliente (Vercel Inc., 2025).

Aspecto	**SSR (Next.js)**	**CSR (React tradicional)**
Onde ocorre a renderização	No servidor, antes do envio ao cliente.	No navegador do usuário.
Entrega inicial da página	HTML completo pré-renderizado.	HTML mínimo + JavaScript que monta a página.
Superfície de ataque principal	Servidor (API Routes, autenticação, acesso a dados).	Cliente (DOM, scripts, armazenamento local).
Riscos comuns	*Broken Access Control*, falhas de configuração, vazamento de dados.	XSS, manipulação do DOM, exposição de tokens.
Impacto na segurança	Maior exposição a ataques no servidor; menor risco no cliente por depender menos de JS.	Maior exposição a ataques no cliente; dependência total de execução JavaScript.

Tabela 1 – Comparação entre pré-renderização no servidor (SSR) e renderização no cliente (CSR) referente à superfície de ataque.

Além disso, temos o *Middleware*, que é crucial para implementar a lógica no lado do servidor, sendo fundamental para a autenticação e os controles de acesso (Vercel Inc., 2025); a Validação de Entrada, que utiliza ferramentas como *Formik* e *Yup Validation* para garantir a integridade dos dados e prevenir ataques de injeção (MUHAMMAD; SYAHPUTRA; ARDIMANSYAH, 2025); e a Proteção de Dados, que se manifesta no uso de bibliotecas como *Nookies* e *bcrypt* para o manuseio seguro e a proteção de informações sensíveis, como senhas (MUHAMMAD; SYAHPUTRA; ARDIMANSYAH, 2025). A implementação correta desses

recursos pode levar a uma redução significativa nas vulnerabilidades (MUHAMMAD; SYAHPUTRA; ARDIMANSYAH, 2025).

2.4 Materiais e Métodos

O método do estudo de caso é baseado na aplicação de técnicas de *hardening* em uma aplicação Next.js, no caso a plataforma Cal.com, e na validação dessas correções através de testes automatizados.

2.4.1 Plataforma Cal.com

O Cal.com é uma plataforma de código aberto projetada para ser uma solução moderna e flexível de agendamento e gestão de eventos, posicionando-se como uma alternativa de alto desempenho que foca em ser extensível, auto-hospedável e amigável ao desenvolvedor. A plataforma utiliza uma arquitetura *full-stack* moderna, sendo construída sobre tecnologias relevantes para a análise de segurança, como o Next.js no *front-end*, que oferece recursos de segurança como o *Server-Side Rendering* (SSR) e o *Middleware* para controle de acesso. No *back-end*, geralmente emprega Node.js com integração a bases de dados como PostgreSQL. Sua natureza *open-source* é um fator duplo: embora permita à comunidade inspecionar o código e corrigir vulnerabilidades mais rapidamente, também expõe o código a atacantes. No contexto do presente trabalho, o Cal.com é a aplicação em teste por ser uma plataforma moderna e amplamente utilizada que manipula dados sensíveis, como horários de eventos e informações de contato. A escolha do Cal.com como objeto de estudo fundamenta-se em três pilares estratégicos: sua representatividade tecnológica como software de produção real em contraste com testes sintéticos, a criticidade dos dados pessoais (PII) que processa, exigindo auditoria rigorosa, e a transparência arquitetural do código aberto, que permite correlacionar achados de testes externos com a implementação interna. Além disso, ressalta-se a alta replicabilidade da pesquisa, uma vez que as configurações de segurança e os vetores de ataque analisados transcendem a plataforma específica, tornando os resultados aplicáveis como guia de referência para a auditoria e proteção de qualquer aplicação desenvolvida no ecossistema Next.js.

2.4.2 Metodologia de Teste com OWASP ZAP

O OWASP ZAP (Zed Attack Proxy) é a ferramenta utilizada para a análise de vulnerabilidades, sendo crucial para a metodologia de teste (OWASP, 2025). O processo segue da seguinte forma, o ZAP envia requisições para a plataforma Cal.com e recebe respostas. Dentre as requisições estão algumas tentativas de exploração das vulnerabilidades. E, por meio das respostas recebidas, o ZAP irá gerar um relatório com os alertas de pontenciais vulnerabilidades.

As técnicas de teste empregadas pelo ZAP incluem o *Ajax Spider*, essencial para mapear o contexto de aplicações modernas (OWASP, 2025), e *Active Scan*, utilizado para realizar tes-

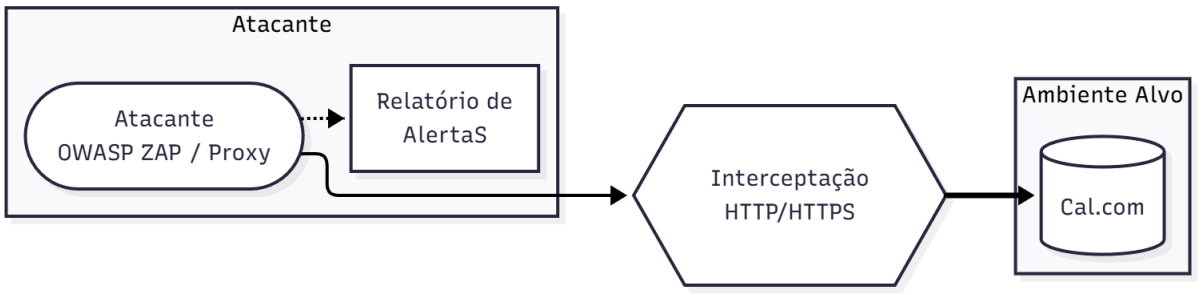

Figura 2 – Cenário de experimentação adotado.

tes de segurança dinâmicos e identificar vulnerabilidades ativas (OWASP, 2025). Além disso, o também ZAP permite classificar os problemas encontrados de acordo com as diretrizes da OWASP.

2.5 Trabalhos Relacionados

O referencial de segurança deste estudo se apoia em pesquisas recentes que analisam a eficácia de *frameworks* modernos e ferramentas automatizadas na detecção e mitigação de riscos. Destaca-se a análise (MUHAMMAD; SYAHPUTRA; ARDIMANSYAH, 2025) sobre a eficácia do Next.js em aprimorar a segurança de aplicações web com base na OWASP Top 10. Este trabalho forneceu *insights* valiosos sobre a minimização de vulnerabilidades através da utilização de recursos nativos do *framework*, como *Server-Side Rendering* (SSR) e *Middleware*, em conjunto com bibliotecas de validação e criptografia, demonstrando uma redução significativa na superfície de ataque da aplicação testada (MUHAMMAD; SYAHPUTRA; ARDIMANSYAH, 2025).

No âmbito das metodologias de detecção, (QUINCOZES et al., 2024), um estudo de caso na plataforma educacional Moodle utilizando a ferramenta OWASP ZAP. A pesquisa identificou uma vasta quantidade de alertas de segurança, evidenciando que instalações padrão de sistemas complexos frequentemente carregam vulnerabilidades críticas, como Injeção de SQL e falhas de configuração, que podem ser mitigadas através de uma análise sistemática baseada nos relatórios da ferramenta (QUINCOZES et al., 2024) De forma similar, temos (CHOIRIYAH; QOMARIASIH, 2023) com uma abordagem híbrida de avaliação de vulnerabilidades e testes de penetração em sites governamentais de saúde na Indonésia. Ao alinhar seus testes com o guia OWASP WSTG v4.2 e a lista Top 10:2021, os autores demonstraram a importância de validar falsos positivos e classificar o risco real das falhas encontradas, como a Quebra de Controle de Acesso (CHOIRIYAH; QOMARIASIH, 2023).

A aplicação de *scanners* automatizados em setores sensíveis também foi explorada em (NURBOJATMIKO et al., 2022), trazendo uma análise da segurança de plataformas de *crowdfunding Sharia*. O estudo reforçou que a categoria de *Security Misconfiguration* é prevalente em aplicações financeiras, exigindo correções imediatas nos cabeçalhos de segurança HTTP e na

gestão de *cookies* para proteger dados de investidores (NURBOJATMIKO et al., 2022). Complementando a visão de análise e detecção, (LALA; KUMAR; T., 2021) foca na perspectiva do desenvolvimento seguro, especificamente utilizando Node.js, a tecnologia subjacente ao Next.js. Os autores propuseram diretrizes práticas de codificação para mitigar vulnerabilidades como injeção de SQL e quebra de autenticação, enfatizando que a segurança deve ser implementada na arquitetura do código, através de *hashing* de senhas e gerenciamento robusto de sessões, e não apenas tratada posteriormente (LALA; KUMAR; T., 2021). Juntos, esses trabalhos fundamentam a necessidade de combinar arquiteturas seguras, como as oferecidas pelo Next.js, com processos rigorosos de verificação automatizada e manual baseados nos padrões da OWASP.

A literatura atual apresenta diversos estudos que aplicam as diretrizes da OWASP e ferramentas de varredura automatizada, como o OWASP ZAP, para a análise de segurança em aplicações web. No entanto, ao analisar criticamente esses trabalhos, observa-se que a maioria se concentra em arquiteturas tradicionais ou utiliza versões anteriores das normas de segurança, deixando lacunas significativas quanto à análise de frameworks modernos de alto desempenho e padrões atualizados.

3 Desenvolvimento

Este capítulo visa mostrar e explicar os métodos utilizados para o diagnóstico da aplicação *Next.js* alvo *Cal.com* e para a proposta de *hardening*. Descreve-se o processo de preparação do ambiente, a execução do scan de vulnerabilidades e a análise de comportamento da aplicação.

3.1 Instalação do Cal.com

Para instalar o *Cal.com* é necessário possuir o `Node 18+`, `Yarn` e um banco de dados `PostgreSQL 13+` (ou, alternativamente, utilizar o `Docker` para subir todo o ambiente por meio do comando `yarn dx`). Primeiramente, deve-se clonar o repositório oficial com:

```
git clone https://github.com/calcom/cal.com.git
```

Em seguida, acesse o diretório do projeto e execute `yarn` para instalar todas as dependências. Para configurar as variáveis de ambiente, crie o arquivo `.env` a partir do modelo `.env.example`. Os segredos `NEXTAUTH_SECRET` e `CALENDSO_ENCRYPTION_KEY` podem ser gerados com o comando:

```
openssl rand -base64 32
```

Após isso, ajuste a variável `DATABASE_URL`, podendo utilizar tanto o banco local quanto o banco iniciado pelo `Docker Compose` ao executar `yarn dx`.

Caso o ambiente seja iniciado via `Docker`, basta possuir `Docker` e `Docker Compose` instalados e executar:

```
yarn dx
```

Isso iniciará automaticamente o banco de dados e os serviços auxiliares. Sem `Docker`, é necessário garantir que o PostgreSQL esteja em execução e, posteriormente, aplicar as migrações utilizando:

```
yarn workspace @calcom/prisma db-migrate
```

Para ambientes de produção, recomenda-se o uso do comando:

```
db-deploy
```

Se desejado, pode-se popular o banco com dados de exemplo utilizando:

```
yarn db-seed
```

A aplicação pode ser iniciada em ambiente de desenvolvimento com:

```
yarn dev
```

(ou `yarn workspace @calcom/web dev`), permitindo o acesso via `http://localhost:3000`. Para ambientes de produção, utilize:

```
yarn build   e  yarn start
```

ou, alternativamente, o comando:

```
docker compose up -d
```

presente no repositório do projeto.

3.2 Execução do Scan de Vulnerabilidades com OWASP ZAP

Após a configuração da aplicação alvo, utiliza-se o OWASP ZAP para realizar o *scan* de vulnerabilidades na aplicação *Next.js*, descrevendo as etapas de *spidering*, análise e ataque ativo executadas pela ferramenta.

3.2.1 Etapas de Análise do ZAP

O processo de teste de segurança dinâmica com o ZAP realiza as seguintes etapas:

1. **Spidering:** O OWASP ZAP realiza o mapeamento da superfície da aplicação. O *Ajax Spider* rastreia as rotas da aplicação *Next.js*, incluindo rotas de API e arquivos JavaScript (*chunks*), garantindo que a ferramenta identifique os *endpoints* disponíveis para teste.
2. **Análise Passiva:** Durante a navegação e interceptação do tráfego, o ZAP analisa os cabeçalhos das respostas HTTP e identifica falhas evidentes, como a ausência de diretivas de segurança (por exemplo, `Content-Security-Policy` e `X-Frame-Options`). Nessa etapa, o relatório inicial registra as vulnerabilidades consideradas críticas.
3. **Ataque Ativo:** O *Active Scan* do ZAP injeta cargas maliciosas e simula ataques nos *endpoints* descobertos. Essa fase tem como objetivo detectar falhas como *injection*, problemas de validação de entrada e possíveis fragilidades em mecanismos de controle de acesso.

Ao concluir todas as etapas, o OWASP ZAP gera um relatório que descreve detalhadamente cada vulnerabilidade ou alerta identificado, acompanhado de recomendações para sua mitigação. O relatório pode ser exportado em diferentes formatos, como Portable Document Format (PDF), HyperText Markup Language (HTML), eXtensible Markup Language (XML) e Comma Separated Values (CSV).

3.3 Proposta de Hardening Documental

A etapa de *hardening* documental consiste na elaboração de uma proposta técnica de mitigação baseada nas vulnerabilidades identificadas pelo OWASP ZAP. Esta fase não foca na alteração direta do código-fonte da aplicação para correção de bugs pontuais, mas sim na reestruturação de configurações arquiteturais permitidas pelo *framework* Next.js para elevar a postura de segurança do sistema de forma global.

O procedimento de *hardening* é conduzido através da correlação entre as falhas de configuração reportadas pelo *scanner* e os recursos nativos de segurança do ecossistema Node.js. O foco da análise recai sobre três pilares fundamentais da estrutura do framework, o arquivo de configuração de cabeçalhos, a camada de *middleware* e os componentes de validação de dados no servidor.

Para as falhas classificadas como configurações incorretas de segurança, a metodologia prevê o uso da função de cabeçalhos no arquivo de configuração do projeto. O objetivo é estabelecer uma política de segurança de conteúdo que restrinja a execução de *scripts* e previna ataques de sequestro de clique, respondendo diretamente aos alertas de ausência de diretivas de proteção nos pacotes HTTP.

No que tange às vulnerabilidades de injeção e falhas de controle de acesso, a estratégia de *hardening* descreve o uso de *middlewares* para a interceptação de requisições. O processo metodológico estabelece que a validação de dados deve ocorrer antes que a lógica de negócio seja processada, utilizando bibliotecas de definição de esquema para garantir que apenas informações devidamente sanitizadas alcancem as rotas de API do servidor.

Dessa forma, a proposta de *hardening* serve como um guia estrutural que conecta o diagnóstico técnico às soluções de engenharia de software. A análise documental permite demonstrar como a superfície de ataque pode ser reduzida através da aplicação correta das funcionalidades de segurança do Next.js, consolidando a defesa da aplicação contra os vetores de ataque simulados durante a fase de scan ativo.

4 Experimentos e Análises

Este capítulo está estruturado em três seções. A primeira aborda a etapa de análise e teste de segurança da aplicação Cal.com, desenvolvida com o framework *Next.js*. A segunda seção apresenta as propostas de correção e mitigação para as vulnerabilidades e alertas identificados durante os testes. Por fim, a terceira seção realiza uma análise comparativa dos riscos, considerando o impacto esperado das soluções propostas.

4.1 Etapa 1 - Teste com a aplicação Cal.com

A Tabela 2 sumariza os alertas reportados pela ferramenta OWASP ZAP para a aplicação Cal.com, classificados em níveis de risco que variam de *Informativo* a *Alto*. Embora a OWASP Top 10:2025 Release Candidate apresente dez categorias distintas, os achados concentram-se predominantemente na categoria A02 – Security Misconfiguration, que abrange desde falhas de cabeçalhos de segurança até exposições de informações do servidor.

Nome da Vulnerabilidade	**Risco**	**Categoria OWASP Top 10:2025**	**Alertas**
Include Server Side	Alto	A05 – Injection	1
Incorrect Configuration Between Domains	Médio	A02 – Security Misconfiguration	1
Content Security Policy (CSP) Header Not Set	Médio	A02 – Security Misconfiguration	13
Missing Anti-clickjacking Header	Médio	A02 – Security Misconfiguration	1
Big Redirect Detected (Potential Sensitive Information Leak)	Baixo	A02 – Security Misconfiguration	23
Cookie No HttpOnly Flag	Baixo	A02 – Security Misconfiguration	7
Date and Time Announcement – Unix	Baixo	A02 – Security Misconfiguration	107
Server Leaks Information via X-Powered-By Header	Baixo	A02 – Security Misconfiguration	14
Private IP Disclosure	Baixo	A02 – Security Misconfiguration	2
X-Content-Type-Options Header Missing	Baixo	A02 – Security Misconfiguration	2
ZAP is Out of Date	Baixo	A09 – Security Logging and Alerting Failures	2
Content-Type Header Missing	Informativo	A02 – Security Misconfiguration	3
Cookie Poisoning	Informativo	A05 – Injection	1
Disclosure of Information – Suspicious Comments	Informativo	A01 – Broken Access Control	70
Information Disclosure – Sensitive Information in URL	Informativo	A01 – Broken Access Control	3
Modern Web Application	Informativo	Nenhuma	9
Session Management Response Identified	Informativo	A07 – Authentication Failures	12
Authentication Request Identified	Informativo	A07 – Authentication Failures	4
User Controllable HTML Element Attribute (Potential XSS)	Informativo	A05 – Injection	2

Tabela 2 – Alertas reportados pela ferramenta OWASP ZAP na aplicação Cal.com. Fonte: Autor (2025).

Além disso, foram identificadas vulnerabilidades críticas e informativas nas categorias A05 – Injection (incluindo *Server Side Include* e *Cookie Poisoning*) e A01 – Broken Access Control, esta última apresentando um volume expressivo de alertas informativos sobre comentários suspeitos e exposição de dados em URLs. O relatório também aponta falhas pontuais em A07 – Authentication Failures e A09 – Security Logging and Alerting Failures, evidenciando que, embora as configurações incorretas sejam o problema mais frequente, a aplicação também apresenta fragilidades em mecanismos de controle de acesso e gestão de sessões.

Note-se que muitos dos alertas reportados são originados da ausência de configurações de segurança opcionais no Next.js que não vêm habilitadas por padrão. O objetivo desta análise consiste em revelar o quão vulnerável uma aplicação moderna pode estar quando o desenvolvedor não implementa medidas de *hardening* específicas do *framework*.

4.1.1 Vulnerabilidades de Risco Alto

Dentre as vulnerabilidades reportadas pela ferramenta OWASP ZAP, destaca-se a identificação de um alerta de Risco Alto: o *Include Server Side*. De acordo com a classificação OWASP Top 10:2025 Release Candidate, esta falha está associada à categoria A05 – Injection.

Em arquiteturas modernas que utilizam Next.js, falhas de injeção no lado do servidor são particularmente críticas, pois podem permitir que um atacante execute scripts ou inclua arquivos maliciosos que serão processados pelo motor de renderização. O relatório indica que essa vulnerabilidade compromete diretamente a integridade do sistema. Como contramedida, é fundamental que nenhuma entrada de usuário seja processada pelo servidor sem uma validação rigorosa e sanitização prévia, impedindo que comandos ou diretivas externas sejam interpretados como parte do código da aplicação.

4.1.2 Vulnerabilidades de Risco Médio

As vulnerabilidades de risco médio identificadas concentram-se na categoria A02 – Security Misconfiguration. Os alertas de *Content Security Policy (CSP) Header Not Set*, *Missing Anti-clickjacking Header* e *Incorrect Configuration Between Domains* evidenciam lacunas na postura defensiva do servidor.

A ausência de uma política de segurança de conteúdo (CSP) é um fator de risco elevado em aplicações Next.js devido à sua natureza de renderização híbrida, permitindo potencialmente a execução de scripts de fontes não autorizadas. Já a falta de cabeçalhos contra *clickjacking* pode expor o usuário a ataques de interface. Para mitigação, recomenda-se a configuração do arquivo `next.config.js` ou do *middleware* da aplicação para injetar cabeçalhos HTTP rígidos, restringindo as origens de recursos e impedindo o carregamento da aplicação em *iframes* de terceiros.

4.1.3 Vulnerabilidades de Risco Baixo

As vulnerabilidades de risco baixo mapeadas envolvem majoritariamente a categoria A02 – Security Misconfiguration, com foco em vazamento de informações e falta de endurecimento (*hardening*) do navegador. Exemplos incluem o *Cookie No HttpOnly Flag*, que facilita o roubo de sessões via scripts, e o vazamento de metadados nos cabeçalhos *X-Powered-By* e nas mensagens de *Date and Time – Unix*.

Também foi identificada a exposição de IPs privados (*Private IP Disclosure*), o que auxilia um atacante na fase de reconhecimento da infraestrutura interna. Adicionalmente, o alerta de *ZAP is Out of Date* foi classificado em A09 – Security Logging and Alerting Failures, ressaltando a importância de manter as ferramentas de auditoria atualizadas para garantir a eficácia da detecção de ameaças.

4.1.4 Alertas Informativos

Os alertas informativos mapeiam comportamentos da aplicação que, embora não representem uma brecha imediata, indicam superfícies de ataque que exigem atenção. Destacam-se as categorias A01 – Broken Access Control, com a identificação de comentários suspeitos no código e informações sensíveis expostas em URLs, e A07 – Authentication Failures, relacionada à identificação de respostas de gerenciamento de sessão e requisições de autenticação.

A presença de *Cookie Poisoning* e atributos HTML controláveis pelo usuário (Potencial XSS) também foram sinalizados como informativos na categoria A05 – Injection. Embora o ZAP os tenha classificado com severidade baixa/informativa neste contexto, eles representam vetores que devem ser higienizados para prevenir a escalada para ataques de Cross-Site Scripting reais.

4.2 Etapa 2 - Propostas de Correção

Esta etapa apresenta as propostas de correção das configurações de segurança para as vulnerabilidades identificadas durante os testes dinâmicos realizados com o OWASP ZAP. Diferentemente de uma abordagem de implementação prática, esta seção adota um caráter analítico e documental, correlacionando os alertas reportados pela ferramenta com soluções arquiteturais e boas práticas recomendadas no contexto do *framework* Next.js.

O objetivo desta etapa é demonstrar como as fragilidades identificadas podem ser mitigadas por meio de ajustes de configuração, validações de entrada e uso adequado de recursos nativos do *framework*, sem a necessidade de detalhar processos de codificação ou depuração.

4.2.1 Análise de Falso Positivo: Server Side Include (SSI)

O alerta classificado como risco alto referente a *Server Side Include* (SSI) foi identificado pelo OWASP ZAP com base em padrões típicos de aplicações que utilizam servidores web tradicionais, como Apache ou Nginx, configurados com suporte à interpretação de diretivas SSI. No entanto, uma análise técnica da arquitetura da aplicação Cal.com indica que essa vulnerabilidade não é explorável no ambiente avaliado.

A aplicação é executada sobre o runtime Node.js, o qual não possui mecanismos nativos para processamento de diretivas SSI. Além disso, a renderização do conteúdo é realizada por meio de componentes React, que aplicam automaticamente o escape de caracteres especiais durante a geração do HTML. Dessa forma, quaisquer sequências que simulem diretivas SSI são tratadas apenas como texto literal, não sendo interpretadas como comandos no lado do servidor. Com base nessa análise, o alerta é caracterizado como um falso positivo no contexto da aplicação avaliada.

4.2.2 Mitigação de Vazamento de Informações e Erros Detalhados

Os alertas relacionados à exposição de informações internas e detalhes de erro indicam a necessidade de reforço na configuração do ambiente de execução. Como proposta de correção, recomenda-se a obrigatoriedade da utilização do modo de produção do framework, por meio da variável de ambiente `NODE_ENV=production`. Essa configuração instrui o Next.js a suprimir mensagens detalhadas de erro e *stack traces* nas respostas HTTP.

Adicionalmente, propõe-se a padronização das respostas das rotas de API por meio de mecanismos de resposta sanitizada, de forma que falhas internas sejam comunicadas apenas através de códigos de status HTTP e mensagens genéricas. A utilização de páginas de erro personalizadas, como `error.tsx` e `not-found.tsx`, complementa essa abordagem ao impedir o vazamento de informações estruturais da aplicação durante falhas de execução.

4.2.3 Segurança de Sessão e Controle de Redirecionamento

Em resposta aos alertas relacionados à gestão de sessão e manipulação de parâmetros de redirecionamento, a proposta de correção envolve a validação rigorosa de URLs de retorno utilizadas nos fluxos de autenticação. Recomenda-se que parâmetros como `callbackUrl` sejam validados exclusivamente contra uma lista de domínios confiáveis definidos em variáveis de ambiente, prevenindo redirecionamentos para origens externas não autorizadas.

O endurecimento da segurança de sessão também inclui a configuração adequada dos cookies de autenticação, com a ativação das flags `HttpOnly` e `Secure`, bem como a definição do atributo `SameSite` como `Lax` ou `Strict`. Essas medidas reduzem significativamente o risco de ataques como *Cross-Site Request Forgery* (CSRF) e fixação de sessão.

4.2.4 Tratamento de Atributos HTML e Prevenção de Injeção

Os alertas informativos e de injeção relacionados ao controle de atributos HTML evidenciam a necessidade de reforço na validação e no tratamento de dados fornecidos pelo usuário. Como proposta de mitigação, recomenda-se que parâmetros sensíveis, como credenciais, não sejam transmitidos por meio de *query strings*, evitando sua exposição em históricos de navegação e logs intermediários.

Embora o React realize automaticamente o escape de caracteres especiais, a adoção de validações de esquema no lado do servidor, utilizando bibliotecas como Zod, contribui para a redução da superfície de ataque. Essa abordagem assegura que apenas dados previamente validados sejam processados ou refletidos na interface, mitigando riscos associados a ataques de injeção e *Cross-Site Scripting* (XSS).

4.2.5 Implementação de Cabeçalhos de Segurança e Política de Conteúdo

Por fim, os alertas relacionados à ausência de cabeçalhos de segurança indicam a necessidade de ajustes nas configurações globais da aplicação. Propõe-se a definição sistemática de cabeçalhos como `X-Content-Type-Options`, `X-Frame-Options` e `Strict-Transport-Security` por meio do arquivo `next.config.js`, garantindo sua aplicação em todas as rotas da aplicação.

A adoção de uma *Content Security Policy* (CSP) em modo restritivo, gerenciada via middleware, é apresentada como uma camada adicional de defesa. Ao restringir a execução de scripts e o carregamento de recursos a domínios explicitamente autorizados, a aplicação reduz significativamente o impacto de possíveis falhas residuais de injeção, fortalecendo sua postura de segurança de forma preventiva e arquitetural.

5 Conclusão

O presente trabalho permitiu analisar de forma sistemática o nível de segurança da aplicação Cal.com, desenvolvida sobre o *framework* Next.js, tomando como referência as diretrizes estabelecidas pela OWASP Top 10:2025 Release Candidate. A partir da execução de testes dinâmicos com a ferramenta OWASP ZAP, foi possível identificar vulnerabilidades, compreender sua origem arquitetural e propor medidas de *hardening* alinhadas às boas práticas de segurança para aplicações web modernas.

Os experimentos realizados demonstraram que, embora o Next.js forneça mecanismos nativos de proteção contra vulnerabilidades clássicas, a configuração padrão da aplicação ainda pode apresentar fragilidades relevantes. Em especial, destacaram-se alertas relacionados à exposição de metadados técnicos, ausência de cabeçalhos de segurança e configurações inadequadas de controle de sessão. Esses achados evidenciam que a segurança não deve ser tratada como uma consequência automática do uso de *frameworks* modernos, mas como um processo contínuo de configuração, validação e endurecimento.

Um dos principais resultados técnicos deste estudo foi a análise aprofundada da vulnerabilidade classificada como Risco Alto relacionada a *Server Side Include* (SSI). A investigação permitiu caracterizar esse alerta como um falso positivo, decorrente das limitações de ferramentas automatizadas em interpretar corretamente arquiteturas baseadas em React e Node.js. Esse resultado reforça a importância da análise crítica dos relatórios de *scanners* de segurança, destacando que a auditoria eficaz depende não apenas da ferramenta utilizada, mas também do conhecimento da arquitetura e do modelo de execução da aplicação avaliada.

As propostas de *hardening* documental apresentadas ao longo do trabalho demonstram que a mitigação de vulnerabilidades pode ser conduzida de forma estrutural, explorando recursos nativos do *framework*, como a centralização de cabeçalhos HTTP, o uso de *middlewares* para aplicação de políticas de segurança de conteúdo e o fortalecimento dos mecanismos de controle de sessão. Essa abordagem evidencia a relevância de deslocar decisões de segurança para o lado do servidor, reduzindo a superfície de ataque exposta ao cliente e aumentando a robustez geral da aplicação.

Conclui-se, portanto, que *frameworks* de renderização híbrida, como o Next.js, representam um avanço significativo no desenvolvimento de aplicações web seguras. Contudo, tais benefícios somente se concretizam quando acompanhados de práticas adequadas de *hardening* e de uma análise criteriosa dos riscos identificados. Este trabalho contribui como um guia metodológico para auditorias de segurança em aplicações semelhantes, demonstrando que a combinação entre ferramentas de análise dinâmica e propostas arquiteturais bem fundamentadas constitui um caminho eficaz para a construção de sistemas web resilientes e confiáveis.

Como trabalhos futuros, propõe-se a utilização de outras ferramentas de varredura de vulnerabilidades, tais como OpenVAS, Nessus e Nikto, com o objetivo de comparar os resultados obtidos com o OWASP ZAP e avaliar o grau de cobertura, precisão e eficiência de cada abordagem no contexto de aplicações desenvolvidas em Next.js. Essa análise comparativa pode auxiliar na definição de uma estratégia de segurança mais abrangente e eficaz.

Adicionalmente, sugere-se a exploração dos logs gerados pelas ferramentas de segurança como conjuntos de dados para técnicas de aprendizado de máquina, visando o desenvolvimento de mecanismos automatizados de detecção de padrões anômalos e comportamentos suspeitos. Tal abordagem pode contribuir para a evolução de sistemas de monitoramento contínuo e resposta a incidentes, ampliando a capacidade de prevenção e mitigação de vulnerabilidades em aplicações web modernas.

Referências

CHOIRIYAH, A.; QOMARIASIH, N. Security analysis on websites belonging to the health service districts in indonesia based on the open web application security project (owasp) top 10 2021. In: *2023 International Conference on Information Technology and Computing (ICITCOM)*. [S.l.: s.n.], 2023. p. 267–272. Citado 4 vezes nas páginas 13, 14, 18 e 25.

KASPERSKY. *KSB Statistics of the Year: The Year in Figures*. [S.l.], 2024. Relatório disponível online. Disponível em: <https://securelist.com/ksb-2024-statistics/114795/>. Citado na página 13.

LALA, S. K.; KUMAR, A.; T., S. Secure web development using owasp guidelines. In: *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*. [S.l.: s.n.], 2021. p. 323–332. Citado 4 vezes nas páginas 12, 13, 17 e 26.

Meta Platforms, Inc. *React Official Documentation*. 2025. <https://react.dev/learn>. Acesso em: 22 out. 2025. Citado na página 12.

MUHAMMAD, R.; SYAHPUTRA, R.; ARDIMANSYAH, M. Website security using next.js: An analytical approach with owasp top 10. *Pertanika Proceedings*, v. 1, 07 2025. Citado 5 vezes nas páginas 13, 18, 23, 24 e 25.

NURBOJATMIKO et al. Security vulnerability analysis of the sharia crowdfunding website using owasp-zap. In: *2022 10th International Conference on Cyber and IT Service Management (CITSM)*. [S.l.: s.n.], 2022. p. 1–5. Citado 6 vezes nas páginas 12, 13, 14, 18, 25 e 26.

OWASP. *The OWASP Top Ten*. 2025. Acesso em: 8 jun. 2025. Disponível em: <https://owasp.org/www-project-top-ten/>. Citado 9 vezes nas páginas 12, 14, 18, 19, 20, 21, 22, 24 e 25.

QUINCOZES, S. et al. Análise de vulnerabilidades da plataforma moodle com base no top 10 da owasp. In: *Anais do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*. Porto Alegre, RS, Brasil: SBC, 2024. p. 739–745. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/sbseg/article/view/30062>. Citado 6 vezes nas páginas 12, 13, 14, 17, 18 e 25.

SRIVASTAVA, S. et al. A comprehensive review of next.js technology: Advancements, features, and applications. *SSRN Electronic Journal*, 01 2024. Citado na página 12.

Vercel Inc. *Next.js Documentation*. [S.l.], 2025. Disponível em: <https://nextjs.org/docs>. Acesso em: 22 out. 2025. Citado 2 vezes nas páginas 12 e 23.