UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA
CURSO BACHARELADO EM INTERDISCIPLINAR EM TECNOLOGIA DA INFORMAÇÃO

THYAGO FABRICIO MELO COSTA

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB DIDÁTICA PARA ESTUDO DE VULNERABILIDADES DE *CROSS-SITE SCRIPTING (XSS)* EM *FRAMEWORKS* MODERNOS

PAU DOS FERROS
2026

THYAGO FABRICIO MELO COSTA

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB DIDÁTICA PARA ESTUDO DE VULNERABILIDADES DE *CROSS-SITE SCRIPTING (XSS)* EM *FRAMEWORKS* MODERNOS

Monografia apresentada à Universidade Federal Rural do Semi-Árido para obtenção do título de Bacharel em Interdisciplinar em Tecnologia da Informação.

Orientador: Walber José Adriano Silva. Prof. Dr.

PAU DOS FERROS

2026

C837d Costa, Thyago Fabricio Melo.
Desenvolvimento de uma Aplicação Web Didática para estudo de Vulnerabilidades de Cross-Site Scripting (XSS) em Frameworks Modernos / Thyago Fabricio Melo Costa. - 2026.
66 f. : il.

Orientador: Walber José Adriano Silva.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Tecnologia da Informação, 2026.

1. Cross-site scripting. 2. React. 3. Desenvolvimento seguro. 4. Segurança de aplicações. 5. Ambiente didático. I. Silva, Walber José Adriano, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Pau dos Ferros
Bibliotecária: Erlanda Maria Lopes da Silva
CRB: 15/966

THYAGO FABRICIO MELO COSTA

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB DIDÁTICA PARA ESTUDO DE VULNERABILIDADES DE *CROSS-SITE SCRIPTING (XSS)* EM *FRAMEWORKS* MODERNOS

Monografia apresentada à Universidade Federal Rural do Semi-Árido para obtenção do título de Bacharel em Interdisciplinar em Tecnologia da Informação.

Defendida em: 24 de Junho de 2026

BANCA EXAMINADORA

Walber José Adriano Silva, Prof. Dr. (UFERSA)
Presidente

Reudismam Rolim de Sousa, Prof. Dr. (UFERSA)
Membro Examinador

Jose Ferdinandy Silva Chagas, Prof. Me. (UFERSA)
Membro Examinador

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, por ser meu sustento e minha força em cada etapa desta caminhada. Sua presença foi fundamental nos momentos de dificuldade e de conquista ao longo desses anos de graduação. À minha família, pelo apoio incondicional, pelo incentivo nos momentos de cansaço e pela confiança depositada em mim durante toda a jornada universitária. Cada palavra de encorajamento e cada gesto de carinho foram essenciais para que eu chegasse até aqui. À minha namorada, Maria Raquel Nunes Silva, por estar ao meu lado nos momentos mais difíceis desta trajetória, pela paciência, pelo cuidado e pelo incentivo constante. Sua presença tornou os dias mais leves e me deu força para continuar quando o caminho parecia pesado. Aos meus amigos, pela companhia ao longo desses anos de graduação, pelas risadas nos intervalos, pelo apoio nas vésperas de prova e por tornarem essa experiência muito mais alegre e marcante. A amizade de vocês foi um dos presentes que a universidade me deu. Ao meu orientador, Prof. Dr. Walber José Adriano Silva, pela dedicação, pela paciência e pelo conhecimento compartilhado ao longo do desenvolvimento deste trabalho. Suas orientações foram fundamentais para o amadurecimento desta pesquisa e para minha formação como profissional. Obrigado por acreditar no potencial deste projeto.

RESUMO

A evolução do desenvolvimento web impulsionou a adoção de arquiteturas baseadas em componentes no lado do cliente, ampliando a superfície de ataque para vulnerabilidades de *Cross-Site Scripting* (XSS). Este trabalho apresenta o desenvolvimento e a validação empírica do SecuriShop, um ecossistema laboratorial didático estruturado como um sistema de *e-commerce* baseado na biblioteca *React* e no servidor *Express*. A plataforma foi projetada com uma arquitetura alternável que permite simular a exploração prática de vetores das categorias Refletido, Armazenado e Baseado em DOM, bem como aplicar e avaliar contramedidas defensivas como Política de Segurança de Conteúdo (CSP), sanitização de dados e *cookies* com a *flag HttpOnly*. A validação técnica foi consolidada por meio de auditorias manuais e automatizadas com a ferramenta XSStrike. Adicionalmente, realizou-se uma validação pedagógica com uma amostra de 38 estudantes de graduação da UFERSA. Os resultados quantitativos demonstraram um impacto didático positivo, evidenciando que a experimentação prática em ambientes controlados contribui significativamente para a quebra de paradigmas sobre a segurança nativa de *frameworks* modernos, mitigando lacunas de formação e estimulando o aprendizado autônomo em desenvolvimento seguro de software.

Palavras-chave: cross-site scripting; react; desenvolvimento seguro; segurança de aplicações; ambiente didático

ABSTRACT

The evolution of web development has driven the adoption of component-based architectures on the client side, expanding the attack surface for Cross-Site Scripting (XSS) vulnerabilities. This work presents the development and empirical validation of SecuriShop, a didactic laboratory ecosystem structured as an e-commerce system based on the React library and the Express server. The platform was designed with a switchable architecture that allows simulating the practical exploitation of vectors from the Reflected, Stored, and DOM-based categories, as well as applying and evaluating defensive countermeasures such as Content Security Policy (CSP), data sanitization, and cookies with the HttpOnly flag. Technical validation was consolidated through manual and automated audits using the XSStrike tool. Additionally, a pedagogical validation was conducted with a sample of 38 undergraduate students from UFERSA. The quantitative results demonstrated a positive didactic impact, showing that hands-on experimentation in controlled environments significantly contributes to breaking paradigms about the native security of modern frameworks, mitigating training gaps, and stimulating autonomous learning in secure software development.

Keywords: cross-site scripting; react; secure development; application security; didactic environment

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

A evolução do desenvolvimento web transformou páginas estáticas em aplicações complexas e altamente interativas, conhecidas como *Single Page Applications (SPAs)*[1], sistemas que carregam e atualizam conteúdos dinamicamente sem a necessidade de recarregar a página inteira. Com esse modelo, *frameworks* de *front-end* como o *React*[2], biblioteca voltada à construção de interfaces baseadas em componentes, passaram a desempenhar um papel central na criação de interfaces dinâmicas, transferindo para o navegador do cliente uma parcela significativa da lógica de renderização e processamento das aplicações (Sangarsu, 2019).

Com o aumento da complexidade das aplicações, a segurança em aplicações web se tornou um aspecto essencial do desenvolvimento. Sob o referencial das diretrizes globais estabelecidas pela *Open Worldwide Application Security Project* (Owasp, 2021), a segurança neste contexto é definida como o conjunto de práticas e mecanismos destinados a proteger sistemas, dados e usuários contra acessos não autorizados, manipulações indevidas e vazamentos de informação, preservando as propriedades de confidencialidade, integridade e disponibilidade (Nieles; Dempsey; Pillitteri, 2017).

Entretanto, essa mudança de modelo também ampliou a superfície de ataque no lado do cliente, pois o cliente passou a ser responsável por parte da lógica do sistema (Hannousse; Yahiouche; Nait-Hamoud, 2022). Entre as ameaças mais recorrentes neste contexto se destaca o *Cross-Site Scripting (XSS)*, uma vulnerabilidade de injeção que permite a execução de *scripts* maliciosos no contexto do navegador da vítima. De acordo com o projeto Owasp (2021), falhas de injeção continuam figurando entre os riscos mais críticos para aplicações web, tendência que permanece evidente nas discussões para a atualização da lista em 2025 (Owasp, 2025), reforçando a persistência desse desafio no cenário da segurança digital.

Embora *frameworks* modernos, como *React*, *Next.js*, *Angular* e *Vue.js*, disponibilizem mecanismos de proteção nativos, a permanência dessas vulnerabilidades frequentemente está associada a fatores humanos. A negligência com boas práticas de segurança, revisões de código superficiais e, principalmente, a falta de compreensão sobre o funcionamento interno das tecnologias por parte dos desenvolvedores resultam em código vulnerável (Xie; Lipford; Chu, 2011). A confiança excessiva nos mecanismos automáticos oferecidos pelos *frameworks*

[1] Aplicações que carregam um único documento HTML inicial e atualizam seu conteúdo dinamicamente por meio de requisições assíncronas, sem a necessidade de recarregar a página a cada interação do usuário.

[2] Biblioteca *JavaScript* de código aberto, mantida pela Meta, voltada à construção de interfaces de usuário baseadas em componentes reutilizáveis. Disponível em: https://react.dev/.

acaba ocultando falhas que seriam evitadas com a adoção consciente de práticas seguras de desenvolvimento.

Um exemplo representativo é a propriedade *dangerouslySetInnerHTML* do *React*. A própria documentação oficial da biblioteca alerta que essa funcionalidade deve ser utilizada com "extrema cautela", pois, caso o desenvolvedor não valide e trate adequadamente os dados inseridos vindos do lado do cliente durante uma navegação, isso torna trivial introduzir uma vulnerabilidade de XSS (META OPEN SOURCE, 2026). Esse risco já foi documentado em casos reais, como a vulnerabilidade XSS identificada no aplicativo de mensagens *Signal*, decorrente do uso inadequado dessa propriedade (De Ryck, 2020). Esse alerta evidencia que a ferramenta, por si só, não é capaz de garantir segurança quando não há um uso criterioso e responsável por parte do programador.

Assim, a principal motivação deste trabalho é investigar, sob uma perspectiva didática, vulnerabilidades que comprometem a integridade dos dados e a privacidade dos usuários em aplicações web, visando compreender seu funcionamento prático em *frameworks* modernos, como o *React*. No cenário atual, informações sensíveis, como *cookies* de sessão e dados de autenticação, constituem alvos constantes de ataques. O XSS, ao possibilitar o sequestro dessas informações, compromete a relação de confiança estabelecida entre o usuário e a plataforma web (OWASP Foundation, 2024). Por isso, este trabalho parte da necessidade de compreender como fortalecer a segurança das aplicações desde a sua concepção, tratando a proteção dos dados dos usuários não como um complemento ao desenvolvimento, mas como parte integrante dele.

Inserir o programador no processo de desenvolvimento seguro é condição fundamental para a construção de sistemas confiáveis (Bertoglio *et al.*, 2022). Sendo assim, é preciso investigar as principais técnicas de invasão de sistemas que comprometem dados sensíveis em aplicações web, e prover um entendimento robusto sobre a dinâmica de seu funcionamento e sobre como os mecanismos de controle e proteção mitigam seus impactos nesses sistemas.

Ainda persiste uma lacuna significativa entre a capacidade de desenvolver uma interface funcional e a competência para protegê-la adequadamente contra ataques de injeção. Essa disparidade é evidenciada por Neto *et al.* (2024), que, com base em pesquisas realizadas pelo SANS Institute e pela Veracode, apontam que apenas 26% dos desenvolvedores consideram possuir habilidades suficientes em segurança cibernética, enquanto 52% das equipes admitem não dispor das competências necessárias para identificar e corrigir vulnerabilidades em seus softwares.

Por fim, este trabalho propõe o desenvolvimento de um ambiente didático de experimentação inspirado em plataformas de treinamento amplamente utilizadas na área de segurança da informação, como o *OWASP Juice Shop*[3]. O ambiente foi construído a partir de um sistema de *e-commerce*, categoria de software amplamente empregada na realização de transações digitais e no processamento de informações sensíveis de clientes, como dados cadastrais e financeiros.

A escolha desse contexto justifica-se pela capacidade de reproduzir cenários próximos à realidade, nos quais vulnerabilidades de segurança podem comprometer funcionalidades essenciais de aplicações web modernas. Contudo, diferentemente do *OWASP Juice Shop*, que funciona como uma plataforma intencionalmente vulnerável voltada à exploração de falhas, a principal contribuição do SecuriShop está em sua arquitetura com alternância entre os modos vulnerável e seguro. Por meio de um sinalizador global no *front-end*, o ambiente foi concebido para não apenas demonstrar, de forma controlada, como as vulnerabilidades podem ser exploradas, mas também permitir a comparação imediata entre os cenários vulnerável e protegido. Essa abordagem favorece o aprendizado prático ao possibilitar que os usuários observem, no mesmo ecossistema e utilizando um *framework* moderno, tanto o comportamento das falhas quanto a aplicação e os efeitos das respectivas estratégias de mitigação.

Por meio da implementação prática de vulnerabilidades de *Cross Site Scripting (XSS)*, a plataforma permite visualizar os impactos desses ataques em um cenário de *e-commerce*, bem como analisar a eficácia das contramedidas aplicadas. Assim, o trabalho contribui para a construção de uma ferramenta didática relevante para o ensino de segurança de software, aproximando conceitos teóricos de situações práticas e favorecendo uma aprendizagem mais efetiva. Pois, o uso de Políticas de Segurança de Conteúdo *(Content Security Policy — CSP)* e o gerenciamento seguro de *cookies* deixam de ser conceitos abstratos e são compreendidos como mecanismos concretos e fundamentais para o fortalecimento da segurança em aplicações modernas web.

1.1 Objetivo Geral

No contexto da segurança no desenvolvimento de aplicações web, que exige conscientização, capacitação e conhecimento técnico por parte dos programadores para a construção de sistemas mais seguros, este trabalho tem como objetivo geral projetar, desenvolver e validar empiricamente um ecossistema laboratorial voltado ao ensino prático de vulnerabilidades de

[3] Disponível em: https://owasp.org/www-project-juice-shop/

Cross Site Scripting (XSS) em *frameworks* modernos. Para alcançar esse objetivo, foi desenvolvida uma aplicação didática com arquitetura alternável entre os modos vulnerável e seguro, permitindo demonstrar tanto a exploração quanto a mitigação dessas vulnerabilidades em um ambiente controlado. Além disso, a plataforma foi submetida a uma validação pedagógica, com o propósito de avaliar sua eficácia como ferramenta de apoio ao ensino e mensurar seu impacto educacional junto a estudantes de graduação.

1.1.1 Objetivos Específicos

Para alcançar o objetivo geral proposto, foram definidos os seguintes objetivos específicos:

- Realizar uma revisão bibliográfica sobre as vulnerabilidades de Injeção (OWASP A03:2021), com foco nos ataques de *Cross-Site Scripting (XSS)*, e os tipos: Refletido, Armazenado e Baseado em DOM, além dos principais impactos e estratégias de mitigação, cujos fundamentos teóricos são apresentados no Capítulo 2;
- Desenvolver uma aplicação web de exemplo, funcional e interativa, utilizando o *framework JavaScript React* para servir como ambiente de estudo de caso, conforme detalhado nos materiais e ambiente de experimentação, descrita na Seção 4.1.
- Implementar de maneira controlada e intencional, as vulnerabilidades de XSS Refletido, XSS Armazenado e XSS Baseado em DOM na aplicação desenvolvida, relacionando cada falha a funcionalidades específicas, como o sistema de busca e a seção de comentários, conforme descrito na Subseção 4.2.1.
- Demonstrar a exploração prática das vulnerabilidades por meio da criação e execução de *payloads* capazes de executar *scripts* no navegador da vítima, evidenciando o impacto real de um ataque como o sequestro de sessão via roubo de *cookies*, conforme validado nas provas de conceito apresentadas na Seção 5.1.
- Aplicar as principais técnicas de mitigação de XSS no código-fonte da aplicação, incluindo o uso de codificação de saída, a implementação de uma Política de Segurança de Conteúdo (*Content Security Policy - CSP*) e a configuração de *cookies* com a flag *HttpOnly*, cujos procedimentos técnicos estão descritos na Subseção 4.2.3.1 e a demonstração da proteção na Seção 5.2.
- Validar pedagogicamente o ambiente laboratorial por meio de sua aplicação em contexto de sala de aula com estudantes de graduação da área de Tecnologia da Informação, coletando dados por meio de um questionário estruturado para analisar sua eficácia como ferramenta

de ensino e os impactos educacionais proporcionados pela experiência, cujos resultados são apresentados e discutidos na Seção 5.3.

- Analisar e validar a eficácia das correções, comparando o comportamento da aplicação antes e depois da implementação das mitigações e confirmando que os *payloads* de ataque foram efetivamente neutralizados, conforme documentado nos relatórios de auditoria das Subseções 5.1.4 e 5.2.4.

1.2 Organização do trabalho

O restante deste trabalho está organizado na seguinte forma: o Capítulo 2 realiza um levantamento teórico sobre conceitos de segurança no desenvolvimento de aplicações web e aborda os trabalhos relacionados na literatura; o Capítulo 3 apresenta a metodologia adotada na pesquisa, detalhando as etapas de fundamentação, desenvolvimento e validação empírica do ecossistema laboratorial; o Capítulo 4 descreve os procedimentos práticos e a engenharia aplicada para construir o ambiente didático utilizando o *React* e o *Express*; o Capítulo 5 apresenta a análise dos resultados obtidos nos ensaios práticos e a discussão da validação pedagógica; e, finalmente, o Capítulo 6 aborda as considerações finais, limitações encontradas e propostas para trabalhos futuros.

2 LEVANTAMENTO TEÓRICO E REVISÃO BIBLIOGRÁFICA

A fundamentação teórica deste trabalho foi desenvolvida por meio de uma revisão bibliográfica sistematizada, com o objetivo de estabelecer o embasamento conceitual sobre vulnerabilidades de injeção e contextualizar o cenário contemporâneo da segurança no desenvolvimento de aplicações web. Para melhor organização do conteúdo, o capítulo encontra-se estruturado em duas seções complementares: a primeira dedica-se à apresentação dos conceitos fundamentais relacionados à infraestrutura de sistemas web e às vulnerabilidades que os afetam; a segunda concentra-se na análise da literatura científica correlata, reunindo estudos e contribuições relevantes para a compreensão do problema investigado.

2.1 Conceitos Sobre Segurança em Aplicações web

2.1.1 Pilares da Fundamentação Teórica

A fundamentação teórica e prática deste estudo está estruturada em três eixos conceituais complementares, que orientam tanto a concepção do ambiente de experimentação quanto a validação dos resultados obtidos.

O primeiro eixo baseia-se nas diretrizes estabelecidas pela Owasp (2021), que classificam as falhas de injeção entre as vulnerabilidades mais críticas presentes em aplicações web modernas. Esse referencial fornece a base conceitual necessária para a compreensão do *Cross-Site Scripting (XSS)* e de suas principais categorias: XSS Refletido, XSS Armazenado e XSS Baseado em DOM. A partir desse entendimento, torna-se possível modelar e reproduzir, de forma controlada, cenários representativos dessas vulnerabilidades, constituindo o núcleo do ambiente laboratorial desenvolvido neste trabalho.

O segundo eixo concentra-se na lacuna existente entre a capacidade de desenvolver software funcional e a competência necessária para garantir sua segurança. Dados apresentados por Neto *et al.* (2024) revelam um cenário preocupante no contexto brasileiro, no qual apenas 26% dos desenvolvedores afirmam possuir conhecimentos suficientes em segurança cibernética, enquanto 52% das equipes de desenvolvimento relatam dificuldades para identificar e corrigir vulnerabilidades em seus próprios sistemas. Esses resultados evidenciam a necessidade de fortalecer a formação prática em segurança de software, aproximando os estudantes de situações reais encontradas no mercado.

Esse desafio torna-se ainda mais relevante diante do cenário tecnológico atual, marcado pela crescente adoção de ferramentas baseadas em Inteligência Artificial no processo de desenvolvimento. Conforme destacado pela (OWASP Foundation, 2025), vulnerabilidades relacionadas ao tratamento inadequado de saídas (*Improper Output Handling*) podem surgir quando desenvolvedores utilizam sugestões geradas por modelos de linguagem sem a devida análise crítica. Nesses casos, códigos inseguros podem ser incorporados às aplicações, introduzindo vulnerabilidades como o XSS. Esse contexto reforça a importância de ambientes educacionais que estimulem a análise crítica, a autonomia técnica e o desenvolvimento de competências voltadas à identificação e mitigação de falhas de segurança.

O terceiro eixo aborda os benefícios técnicos e econômicos da prevenção de vulnerabilidades ainda nas fases iniciais do ciclo de desenvolvimento de software. De acordo com Neto *et al.* (2024), corrigir falhas durante a etapa de implementação apresenta custos significativamente menores quando comparado às intervenções realizadas após a implantação do sistema em produção. Essa diferença evidencia a importância da adoção de práticas seguras desde o início do processo de desenvolvimento, reforçando a necessidade de preparar futuros profissionais para incorporar princípios de segurança de forma contínua em seus projetos.

2.1.2 Fundamentos e Arquiteturas Modernas

Para compreender a superfície de exploração das vulnerabilidades presentes nas aplicações web modernas, é necessário analisar a evolução da arquitetura desses sistemas ao longo do tempo. Inicialmente, a web operava com base em um modelo estático de requisição e resposta, no qual o navegador enviava uma solicitação HTTP (*Hypertext Transfer Protocol*) ao servidor, que processava a requisição e retornava um documento HTML (*Hypertext Markup Language*) completo para ser exibido ao usuário.

Com o aumento da demanda por aplicações mais interativas e dinâmicas, a arquitetura web passou a ser organizada em duas camadas principais: o *back-end* e o *front-end* (Li *et al.*, 2024). O *back-end* corresponde ao lado do servidor e é responsável pelo processamento das regras de negócio, pela persistência de dados em bancos de dados, pela disponibilização de APIs (*Application Programming Interfaces*) e pelo gerenciamento de autenticação e sessões. Já o *front-end* representa o lado do cliente, abrangendo a interface gráfica, as folhas de estilo (CSS), os elementos do DOM (*Document Object Model*) e os *scripts JavaScript* executados diretamente no navegador.

Essa evolução arquitetural contribuiu para a popularização das *Single Page Applications (SPAs)*. Diferentemente das aplicações tradicionais baseadas em múltiplas páginas, as SPAs carregam inicialmente apenas um documento HTML, realizando a atualização de conteúdo e a navegação entre rotas de forma dinâmica e assíncrona. Nesse modelo, os dados, geralmente estruturados em formato JSON *(JavaScript Object Notation)*, são obtidos por meio de chamadas a APIs e renderizados no navegador por bibliotecas e *frameworks* como *React* e *Next.js*. Embora essa abordagem proporcione melhor experiência de uso e maior desempenho, ela também transfere uma parcela significativa da lógica de processamento e manipulação de dados para o ambiente do cliente, ampliando consideravelmente a superfície de ataque no lado do cliente. Essa descentralização arquitetural cria novos desafios de segurança no desenvolvimento de software. Nesse sentido, conforme apontam Queiroz e Cavalcanti (2024), observa-se uma lacuna preocupante na formação acadêmica em tecnologia, uma vez que a maioria dos graduados ingressa no mercado sem o preparo técnico necessário para realizar testes de segurança e proteger adequadamente as interfaces modernas.

2.1.3 Conceitos de Segurança de Software e Taxonomia do *Cross-Site Scripting (XSS)*

No contexto de ciclo de vida de software, uma vulnerabilidade pode ser compreendida como uma fraqueza, falha de projeto ou erro de implementação passível de exploração por agentes maliciosos para comprometer a segurança de um sistema. Conforme estabelecido pelo *Application Security Verification Standard (ASVS)* da OWASP Foundation (2020), o desenvolvimento seguro atua de forma preventiva, incorporando práticas, diretrizes e mecanismos de verificação desde as fases iniciais de concepção até a implantação em produção. Essa abordagem visa reduzir a ocorrência de falhas críticas e os custos associados à sua correção em etapas posteriores.

Entre as vulnerabilidades mais relevantes presentes em aplicações web modernas destaca-se o *Cross-Site Scripting (XSS)*, especialmente em arquiteturas que concentram parte significativa do processamento no lado do cliente. Segundo o projeto Owasp (2021), essa vulnerabilidade ocorre quando dados fornecidos por usuários são processados e exibidos sem os mecanismos adequados de validação, sanitização ou codificação, permitindo que o navegador interprete conteúdo malicioso como código executável. Como consequência, *scripts* arbitrários podem ser executados no contexto da sessão da vítima, possibilitando diferentes formas de comprometimento da aplicação e dos dados manipulados.

Tradicionalmente, o XSS é classificado em três categorias principais. O XSS Refletido (*Reflected XSS*) ocorre quando o código malicioso é enviado ao servidor por meio de parâmetros da requisição, como campos de busca ou valores presentes na URL, sendo imediatamente retornado na resposta e executado no navegador da vítima, sem qualquer persistência. O XSS Armazenado (*Stored XSS*) caracteriza-se pelo armazenamento permanente do *payload* malicioso em componentes da aplicação, como bancos de dados, formulários ou áreas de comentários. Nesses casos, o código injetado é executado automaticamente sempre que o conteúdo comprometido é acessado por outros usuários. Já o XSS Baseado em DOM (*DOM-based XSS*), conforme descrito por Klein (2005), ocorre exclusivamente no lado do cliente, quando dados controlados pelo usuário são manipulados por *scripts JavaScript* e inseridos de forma insegura na árvore do DOM (*Document Object Model*), sem participação direta do servidor no processamento da carga maliciosa.

Em resposta aos riscos associados a esse tipo de vulnerabilidade, surgiram diversas plataformas voltadas ao ensino prático de segurança de aplicações. Ferramentas como o *OWASP Juice Shop* foram desenvolvidas para oferecer ambientes controlados e intencionalmente vulneráveis, nos quais estudantes e profissionais podem explorar cenários realistas de ataque e defesa sem comprometer sistemas em produção. A utilização dessas plataformas complementa o ensino teórico tradicional e, conforme destacam Queiroz e Cavalcanti (2024), representa uma estratégia relevante para reduzir a carência de laboratórios práticos nos cursos de tecnologia, contribuindo para a formação de profissionais mais preparados para os desafios do desenvolvimento seguro de software.

2.1.4 Estratégias de Mitigação de XSS na Literatura

A mitigação eficaz de vulnerabilidades XSS em arquiteturas web modernas baseia-se na adoção de uma estratégia de defesa em camadas, que combina mecanismos de proteção no código-fonte com políticas de segurança aplicadas pelo navegador. Nessa direção, as recomendações do consórcio internacional *World Wide Web Consortium* (W3C, 2026) destacam a Política de Segurança de Conteúdo (*Content Security Policy — CSP*) como uma das principais medidas para reduzir os riscos associados à execução de código malicioso. Por meio da definição de regras que restringem as origens permitidas para carregamento de recursos e bloqueiam a execução de *scripts inline*, a CSP atua como uma camada adicional de proteção, capaz de mitigar os impactos de tentativas de injeção mesmo quando conteúdos maliciosos alcançam a interface da aplicação.

Além das barreiras técnicas, estudos sobre o desenvolvimento de aplicações baseadas em componentes apontam que a confiança excessiva em mecanismos automatizados pode representar um fator de risco. Ao abordar o uso de propriedades de renderização, como o *dangerouslySetInnerHTML*, a documentação oficial do *React*, mantida pela META OPEN SOURCE (2026), ressalta que recursos desse tipo transferem ao desenvolvedor a responsabilidade pelo tratamento seguro dos dados exibidos na interface. Na ausência de práticas adequadas de codificação de saída e sanitização contextual, o uso inadequado dessas funcionalidades pode facilitar a introdução de vulnerabilidades XSS na aplicação.

Diante desse cenário, diversos pesquisadores defendem que as contramedidas técnicas devem ser acompanhadas por iniciativas de formação em desenvolvimento seguro, capazes de aproximar os conceitos teóricos das demandas da prática profissional. Esse cenário evidencia a importância de ferramentas educacionais e ambientes de simulação controlada como estratégias complementares à formação acadêmica, contribuindo para a preparação de desenvolvedores mais aptos a identificar e corrigir falhas de segurança no mercado de trabalho (Bertoglio *et al.*, 2022).

2.2 Revisão da literatura

A fundamentação teórica deste estudo baseia-se em uma revisão bibliográfica estruturada, realizada a partir da consulta a artigos científicos e relatórios técnicos recentes disponíveis em repositórios de referência, como o Google Acadêmico [4] e a SBC OpenLib (SOL)[5]. A estratégia de busca concentrou-se na identificação de pesquisas nacionais que relacionam o ensino de Engenharia de Software às práticas de desenvolvimento seguro, à análise de vulnerabilidades em aplicações web e ao uso de ambientes didáticos para simulação controlada de cenários de ataque e defesa.

A seção a seguir apresenta e sintetiza os quatro estudos que constituem o principal suporte teórico desta pesquisa, contribuindo para fundamentar, delimitar e contextualizar a proposta desenvolvida.

2.2.1 Estratégia de Busca e Bases de Dados

Para garantir a relevância e a confiabilidade das fontes, as buscas foram conduzidas com descritores como ’vulnerabilidades de software’, ’*Cross-Site Scripting (XSS)*’, ’segurança em

[4] Disponível em: https://scholar.google.com/?hl=pt-BR
[5] Disponível em: https://sol.sbc.org.br/index.php/indice

frameworks modernos' e 'lacuna de conhecimento em segurança'. Como critério de temporalidade, priorizaram-se publicações entre 2021 e 2025, assegurando alinhamento com o contexto tecnológico atual.

Quadro 1 – Síntese dos trabalhos relacionados

Referência	**Escopo**	**Vulnerabilidades estudadas**	**Ferramentas utilizadas**	**Contribuição para o trabalho**
Neto *et al.* (2024)	Avaliação da maturidade de equipes e lacunas na formação de profissionais no mercado nacional.	Injeção geral, falhas de lógica e diretrizes gerais do OWASP Top 10.	Questionários estruturados e análise de relatórios industriais da área de tecnologia.	Comprova estatisticamente a carência de competências defensivas em desenvolvedores brasileiros, justificando a criação de plataformas instrucionais funcionais.
Queiroz e Cavalcanti (2024)	Investigação sobre o impacto de ferramentas no suporte pedagógico e metodológico do ensino prático de computação.	Segurança da informação e rotinas defensivas aplicadas ao ecossistema web.	Plataformas de estudo de caso de desenvolvimento e questionários instrucionais.	Apoia a modelagem empírica e evidencia a importância dos laboratórios práticos na mitigação de lacunas técnicas de alunos de graduação.
Bertoglio *et al.* (2022)	Relato de experiência sobre uma metodologia de construção de conhecimento em Segurança Ofensiva baseada em comunidades de prática (Grupo *Weasels*).	Enumeração web, falhas em serviços de rede, injeções de código e escalação de privilégios.	Laboratórios práticos virtuais e plataformas online de treinamento (*TryHackMe*).	Fundamenta o uso de laboratórios interativos guiados e cooperativos, demonstrando que a experimentação prática gera maior evolução técnica no contexto web.
Rosa *et al.* (2024)	Análise empírica e comparativa da capacidade de cobertura e eficácia de ferramentas de varredura automatizada web.	Injeções de código, *Cross-Site Scripting* (XSS), falhas de autenticação e configurações de servidores.	Scanners automatizados (*OWASP ZAP, Nuclei, Nikto, Wapiti*) testados sobre o ecossistema *Juice Shop*.	Sustenta cientificamente a escolha de auditorias automatizadas para testar se as protections inseridas no código neutralizam os vetores críticos do OWASP Top 10.

Fonte: Autoria própria (2026)

Ao analisar os trabalhos relacionados apresentados, observa-se uma convergência entre

as pesquisas recentes quanto à importância de associar conceitos teóricos a experiências práticas no ensino de desenvolvimento seguro. Nesse contexto, o estudo de Neto *et al.* (2024) reforça a relevância educacional desta pesquisa ao evidenciar a carência de competências práticas em segurança digital entre profissionais e equipes de desenvolvimento de software no cenário brasileiro.

No campo das abordagens pedagógicas, Bertoglio *et al.* (2022) e Queiroz e Cavalcanti (2024) demonstram que metodologias baseadas em laboratórios práticos e na resolução de desafios favorecem significativamente o processo de aprendizagem quando comparadas a estratégias exclusivamente expositivas. Os resultados desses estudos indicam que a interação direta com ambientes de experimentação contribui para uma melhor assimilação dos conceitos de segurança e para o desenvolvimento de habilidades técnicas aplicadas.

Por sua vez, Rosa *et al.* (2024) destacam a importância das auditorias automatizadas como mecanismo de validação técnica em aplicações web. Os autores demonstram que a análise contínua de código em ambientes reconhecidos internacionalmente, como o *OWASP Juice Shop*, fornece indicadores relevantes para avaliar a eficácia de mecanismos de proteção e identificar vulnerabilidades de forma sistemática. Essa perspectiva dialoga diretamente com a metodologia adotada neste trabalho, que combina testes práticos e validação automatizada para verificar a efetividade das contramedidas implementadas na plataforma desenvolvida.

3 METODOLOGIA

Este trabalho caracteriza-se como uma pesquisa aplicada, de abordagem qualitativa e caráter exploratório, pois busca compreender e demonstrar o funcionamento das vulnerabilidades do tipo *Cross-Site Scripting (XSS)*, bem como as técnicas empregadas para sua prevenção e mitigação. Para atingir esse objetivo, foi desenvolvido e analisado um sistema de comércio eletrônico (*e-commerce*), escolhido por representar uma categoria de aplicação amplamente utilizada no processamento de dados dinâmicos e sensíveis, além de reunir funcionalidades que favorecem a modelagem e a demonstração prática de cenários de exploração de vulnerabilidades em aplicações web.

A metodologia adotada estrutura-se em três etapas complementares. Na primeira, realizou-se uma revisão da literatura especializada sobre vulnerabilidades de injeção, com ênfase em *Cross-Site Scripting (XSS)*, a partir de documentos da OWASP *Foundation*, artigos científicos e materiais técnicos reconhecidos na área. Na segunda, desenvolveu-se uma aplicação de comércio eletrônico utilizando a biblioteca *React*, com o objetivo de criar um ambiente controlado para experimentação e análise. Nesse contexto, foram implementadas intencionalmente vulnerabilidades das categorias XSS Refletido, XSS Armazenado e XSS Baseado em DOM, associadas a funcionalidades específicas da plataforma, como o mecanismo de busca e o sistema de comentários e, por fim, consolidando-se na validação pedagógica em ambiente de instrução para mensurar a eficácia da plataforma junto aos estudantes.

Para alcançar o propósito estabelecido na etapa de desenvolvimento, a aplicação web didática foi estruturada sob o modelo de um sistema de comércio eletrônico (*e-commerce*). A escolha desse tipo de aplicação justifica-se pela diversidade de funcionalidades e fluxos de dados que oferece, incluindo mecanismos de busca, áreas de comentários persistentes e manipulação de parâmetros na interface, proporcionando um cenário realista e aplicável, buscando facilitar a compreensão através de demonstração prática de vulnerabilidades e vetores de ataque, que são meios, métodos e caminhos utilizados por atacantes visando obter acesso não autorizado ao sistema. Em contrapartida, a adoção de um ambiente de *e-commerce* também traz desafios adicionais ao desenvolvimento do projeto, uma vez que a complexidade associada à arquitetura da aplicação e ao gerenciamento de estado no *front-end* demanda uma atenção maior de modelagem e implementação para que as vulnerabilidades possam ser reproduzidas de forma controlada e didática, sem comprometer a estabilidade e o funcionamento do fluxo comercial simulado.

A partir dessa estrutura, foram elaborados e executados diferentes *payloads* para demons-

trar, na prática, os impactos dessas vulnerabilidades, incluindo a execução de código malicioso no navegador do usuário e o comprometimento de informações de sessão. Por fim, foram aplicadas contramedidas diretamente no código-fonte da aplicação, contemplando técnicas como codificação de saída, implementação da Política de Segurança de Conteúdo (*Content Security Policy – CSP*) e configuração adequada de *cookies* de autenticação com a propriedade *HttpOnly*

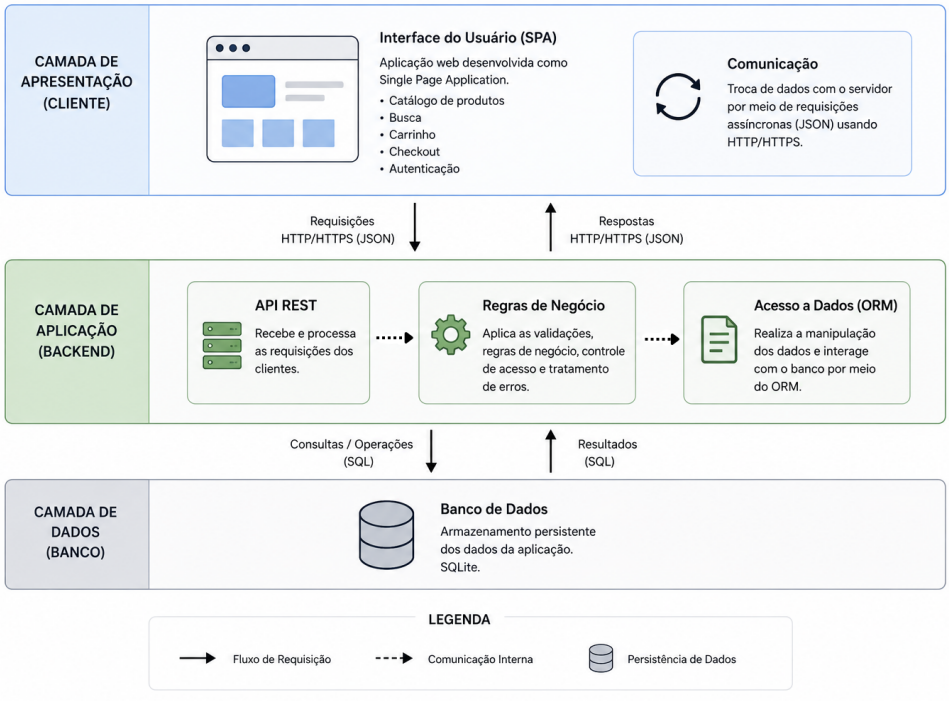
Como etapa essencial para a consolidação e validação do ecossistema proposto, a pesquisa contemplou uma fase de validação empírica e pedagógica, na qual a aplicação desenvolvida foi utilizada em sala de aula como ferramenta de apoio ao ensino de segurança em aplicações web. Nessa atividade, estudantes de graduação interagiram diretamente com os cenários vulneráveis implementados na plataforma, explorando, de forma prática, os impactos das falhas de segurança e os mecanismos empregados para mitigá-las. Ao final da experiência, os participantes responderam a um questionário estruturado e anônimo, cujos dados subsidiaram a análise da usabilidade da ferramenta, de sua contribuição para o processo de ensino e aprendizagem e de sua eficácia como instrumento de conscientização sobre vulnerabilidades e práticas seguras de desenvolvimento de *software*.

4 DESENVOLVIMENTO

Este capítulo apresenta os aspectos práticos relacionados à concepção e ao desenvolvimento do ecossistema didático de comércio eletrônico (*e-commerce*), detalhando a infraestrutura empregada, a *stack* tecnológica adotada e a especificação metodológica das rotas da API utilizadas nos experimentos. Para facilitar a compreensão dos fluxos de dados e das estratégias de mitigação avaliadas, a arquitetura da aplicação foi estruturada em camadas, permitindo delimitar com clareza os pontos de exposição associados aos vetores de ataque analisados.

Para facilitar a compreensão da organização e da interação entre os componentes do ecossistema desenvolvido, a infraestrutura foi representada por meio de um diagrama arquitetural baseado no modelo clássico de três camadas. Apresentado na Figura 1, esse modelo ilustra o fluxo de comunicação do sistema e permite visualizar como as requisições são processadas entre o lado do cliente e o lado do servidor. Além disso, o diagrama evidencia a distribuição das responsabilidades entre as camadas da aplicação, servindo como base para a compreensão dos mecanismos de processamento, acesso a dados e execução das funcionalidades. Tanto os cenários vulneráveis quanto os mecanismos de proteção implementados neste trabalho foram desenvolvidos de forma integrada sobre essa mesma arquitetura.

Figura 1 – Diagrama da arquitetura do ecossistema em camadas

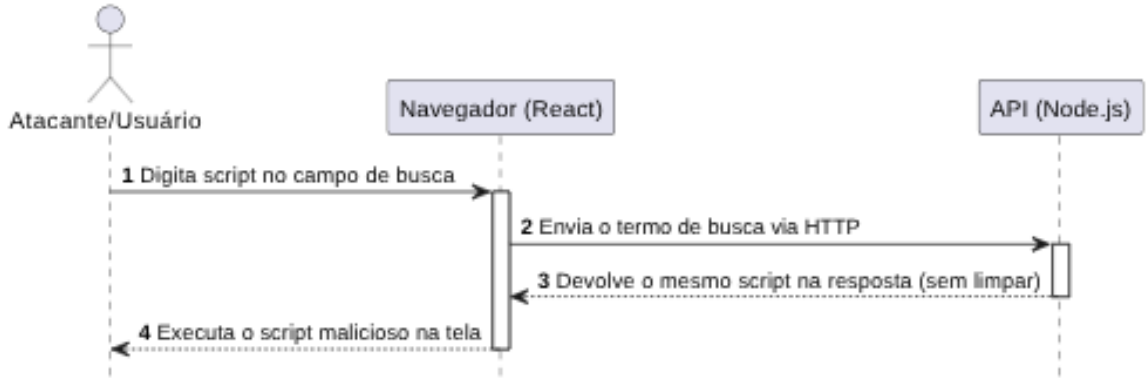


Fonte: Autoria própria (2026)

A aplicação foi desenvolvida com base em uma arquitetura de três camadas, na qual cada camada possui responsabilidades específicas. A interface do usuário foi implementada como uma *Single Page Application (SPA)* utilizando *React*, comunicando-se de forma assíncrona com uma API desenvolvida em *Node.js*. Essa API é responsável pelo processamento das requisições e pelo gerenciamento da persistência dos dados em um banco *SQLite* por meio do *Prisma ORM*.[6] Diante disso, as vulnerabilidades analisadas foram intencionalmente incorporadas a diferentes pontos de manipulação de dados, permitindo a reprodução controlada dos principais vetores de ataque do XSS. Os respectivos fluxos de exploração são apresentados a seguir.

O primeiro vetor modelado corresponde ao XSS Refletido (*Reflected XSS*), implementado no mecanismo de busca de produtos da aplicação. Nesse cenário, os dados inseridos pelo usuário são processados e retornados pelo servidor sem o tratamento adequado, possibilitando a execução de conteúdo malicioso na interface. Conforme ilustrado na Figura 2, o fluxo de exploração caracteriza-se pelo envio do termo de busca ao servidor e pela sua devolução imediata ao cliente, evidenciando o ciclo de requisição e resposta que caracteriza essa categoria de vulnerabilidade.

Figura 2 – Diagrama de sequência do fluxo de ataque para XSS Refletido



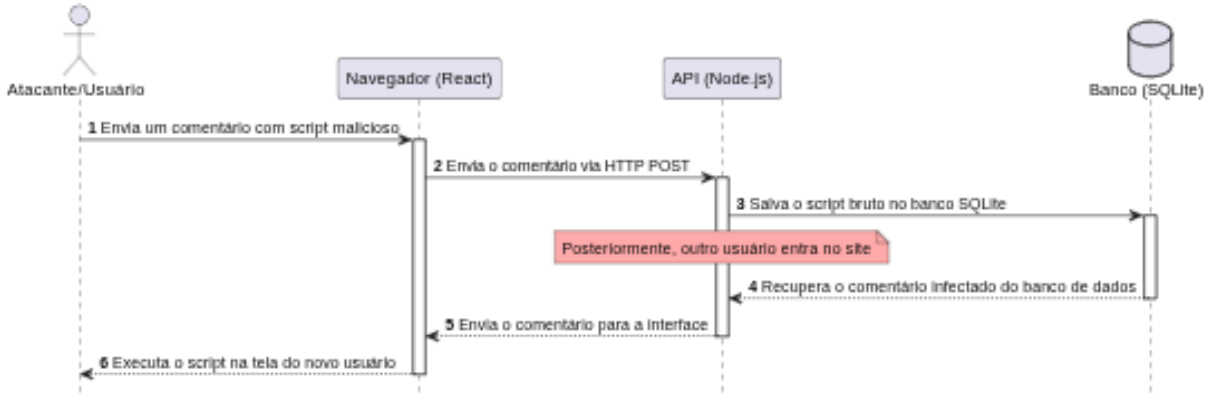
Fonte: Autoria própria (2026)

O segundo vetor modelado corresponde ao XSS Armazenado (*Stored XSS*), implementado na funcionalidade de comentários e avaliações da aplicação. Nesse tipo de vulnerabilidade, o conteúdo malicioso é persistido na camada de armazenamento de dados e passa a fazer parte das informações disponibilizadas pelo sistema. Conforme ilustrado na Figura 3, o fluxo de exploração ocorre em duas etapas distintas. Inicialmente, o atacante submete o código malicioso por meio da interface da aplicação, permitindo que ele seja armazenado no banco de dados *SQLite*. Em um momento posterior, esse conteúdo é recuperado e exibido a outros usuários

[6] Disponível em: https://www.prisma.io/orm

durante a navegação, ocasionando a execução automática do código injetado no navegador das vítimas.

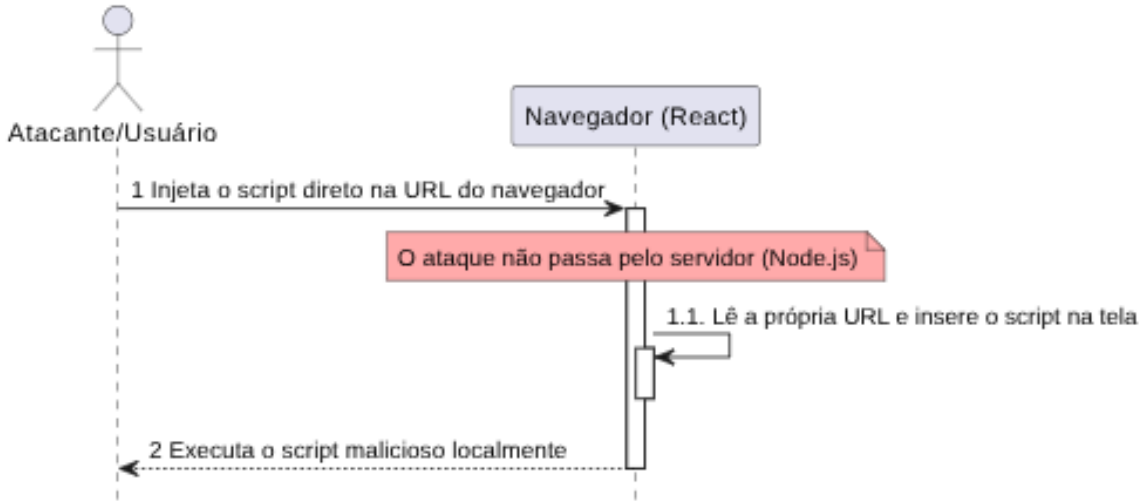
Figura 3 – Diagrama de sequência do fluxo de ataque para XSS Armazenado



Fonte: Autoria própria (2026)

Por fim, o terceiro vetor corresponde ao XSS Baseado em DOM (*DOM-based XSS*), implementado a partir da manipulação de dados presentes na URL e processados diretamente pelo navegador. Conforme ilustrado na Figura 4, esse tipo de vulnerabilidade ocorre exclusivamente no lado do cliente, sem depender da participação do servidor durante a execução do ataque. Nesse cenário, parâmetros controlados pelo usuário são extraídos da *query string* da URL e processados pela aplicação *React* de forma insegura, resultando na renderização e execução local do conteúdo malicioso. Desse modo, a exploração ocorre inteiramente no ambiente do navegador, sem a necessidade de novas requisições, processamento adicional ou interação direta com o servidor *Node.js*.

Figura 4 – Diagrama de sequência do fluxo de ataque para XSS Baseado em DOM



Fonte: Autoria própria (2026)

4.1 Materiais e Ambiente de Experimentação

Para garantir a reprodutibilidade dos testes e o isolamento controlado das vulnerabilidades, foi configurado um ambiente de desenvolvimento padronizado, utilizando ferramentas capazes de simular o ecossistema real de aplicações web modernas.

4.1.1 Infraestrutura do Desenvolvimento da Ferramenta: *hardware* e *software*

O desenvolvimento do *e-commerce* e a execução dos ataques controlados foram realizados em ambiente local, com as seguintes especificações:

- ***Hardware*:** *laptop* Acer Nitro 5 (AN515-58), equipado com recursos de alto desempenho, capazes de suportar a execução simultânea do ambiente de desenvolvimento, banco de dados e ferramentas de análise.
- **Sistema Operacional:** distribuição Linux Pop!_OS, escolhida por sua estabilidade e pelo gerenciamento eficiente de dependências voltadas ao desenvolvimento.
- **Ambiente de Execução:** utilização do *Node.js*, em conjunto com o gerenciador de pacotes *npm*, para o gerenciamento das bibliotecas e dependências do projeto.

4.1.2 *Stack* Tecnológica e Arquitetura

A aplicação de *e-commerce* foi construída com base em uma *stack* tecnológica amplamente adotada no mercado, o que confere maior aderência aos cenários encontrados no

desenvolvimento profissional.

- ***Front-end:*** utilização da biblioteca *React*, integrada à ferramenta de *build Vite*, permitindo a construção de uma interface modular e de alto desempenho.
- **Arquitetura de Interface:** adoção da metodologia *Atomic Design*, que organiza os componentes em níveis como átomos, moléculas e organismos. Essa abordagem possibilita o isolamento de falhas de segurança em partes específicas da interface, como o campo de busca de produtos, de forma mais precisa.
- **Persistência de Dados:** utilização do Prisma ORM para comunicação com um banco de dados SQLite, garantindo uma estrutura leve para o armazenamento de dados, como avaliações de clientes, que foram exploradas em testes de vulnerabilidades do tipo *Stored XSS*.

4.2 Procedimentos Experimentais

O desenvolvimento prático deste trabalho seguiu um ciclo estruturado em três fases distintas: modelagem da vulnerabilidade, exploração e mitigação.

4.2.1 Modelagem da Vulnerabilidade

Nesta etapa, as funcionalidades da aplicação de *e-commerce* foram configuradas intencionalmente para permitir a inserção e a execução de códigos maliciosos, simulando falhas de implementação para fins de análise e demonstração. Diferentemente da visão arquitetural apresentada anteriormente, esta fase concentrou-se na modelagem prática de três cenários específicos de vulnerabilidade, nos quais os vetores de injeção foram incorporados diretamente às regras de manipulação de dados do sistema.

O primeiro cenário corresponde ao XSS Refletido, implementado no mecanismo de busca de produtos da plataforma. Nesse fluxo, o componente de *front-end* captura os termos informados pelo usuário por meio de parâmetros presentes na URL e os encaminha para a API. No lado do servidor, a rota responsável pelo processamento da consulta foi configurada para receber e retornar esses dados diretamente na resposta, sem qualquer tratamento ou codificação de caracteres especiais. Como consequência, conteúdos potencialmente maliciosos podem ser interpretados pelo navegador durante a renderização da página de resultados, reproduzindo o comportamento típico dessa categoria de vulnerabilidade.

O segundo cenário contempla o XSS Armazenado, implementado na funcionalidade de comentários e avaliações dos produtos. Para simular a persistência do conteúdo malicioso, a rota de cadastro foi desenvolvida de forma a armazenar as informações recebidas diretamente no banco de dados *SQLite* por meio do Prisma ORM, sem a aplicação de mecanismos de validação ou sanitização. Dessa forma, qualquer código inserido permanece registrado na base de dados e é posteriormente recuperado pela aplicação, sendo exibido e executado sempre que outros usuários acessam o conteúdo comprometido.

O terceiro cenário aborda o XSS Baseado em DOM, cuja exploração ocorre exclusivamente no lado do cliente. Diferentemente dos casos anteriores, a vulnerabilidade foi implementada a partir da captura e manipulação insegura de parâmetros da *query string* da URL diretamente pela interface desenvolvida em *React*. Nesse fluxo, os dados extraídos localmente são encaminhados para os pontos de renderização dinâmica no *front-end* sem passar por qualquer tratamento prévio no servidor *Node.js*, forçando a execução do código malicioso no navegador. Essa abordagem evidencia os riscos associados à manipulação inadequada de elementos do *Document Object Model (DOM)* e à ausência de validações contextuais na camada de apresentação.

Durante o desenvolvimento desses cenários, observou-se que a propriedade *dangerouslySetInnerHTML*, disponibilizada pelo *React*, segue as diretrizes de segurança adotadas pelos navegadores e, por padrão, não executa automaticamente elementos <*script*> inseridos por meio de *innerHTML*. Ainda assim, a própria documentação oficial da biblioteca alerta que seu uso exige cautela, uma vez que a renderização de conteúdo não confiável sem tratamento adequado pode resultar na introdução de vulnerabilidades de *Cross-Site Scripting (XSS)*.

Com o objetivo de reproduzir esse cenário de uso inadequado em um contexto controlado, foi desenvolvido um componente customizado capaz de simular a utilização insegura dessa funcionalidade, sem qualquer processo de sanitização dos dados recebidos. O componente foi projetado para contornar as restrições nativas de execução e permitir o processamento dos *payloads* de teste diretamente no navegador, viabilizando a demonstração prática dos ataques implementados.

Essa decisão de projeto possui caráter estritamente didático e busca reproduzir situações reais decorrentes de erros de implementação frequentemente observados no desenvolvimento de aplicações web modernas. Dessa forma, o ambiente laboratorial permite evidenciar, de maneira prática e controlada, os riscos associados ao tratamento inadequado de dados dinâmicos na interface do usuário e os impactos que essas falhas podem gerar na segurança da aplicação.

4.2.2 Execução de Ataques e Demonstração de Impacto

Com o ambiente vulnerável devidamente configurado, procedeu-se à criação e execução de *payloads* em *JavaScript* para validação das falhas. Os testes concentraram-se nos vetores principais:

- **Execução Local:** utilização de um *script* para alterar propriedades de estilo do DOM (como fundo escuro e texto em destaque) e modificar o título da página para indicar comprometimento, evidenciando o controle sobre a interface do usuário.
- **Sequestro de Sessão:** implementação de um *script* capaz de acessar e exibir informações de sessão armazenadas no navegador, demonstrando o risco de exposição de identificadores sensíveis e reforçando os impactos críticos desse tipo de vulnerabilidade.
- **Manipulação de Contexto (DOM):** a validação da falha foi realizada por meio da injeção de *payloads* na *query string* da URL. O procedimento comprovou que a lógica de *front-end* capturou e executou o *script* malicioso de forma independente do servidor, caracterizando a vulnerabilidade no lado do cliente.
- **Auditoria automatizada de vetores (XSStrike):** utilizou-se a ferramenta XSStrike[7] para executar testes de *fuzzing* inteligente nos parâmetros de entrada da aplicação operando em modo vulnerável. A ferramenta foi responsável por injetar múltiplos vetores complexos de forma automatizada, permitindo mapear com precisão os pontos de falha e confirmar a viabilidade de exploração das vulnerabilidades refletidas e baseadas em DOM.

4.2.3 Implementação de Mitigações e Validação

Para neutralizar as vulnerabilidades modeladas e consolidar os objetivos deste trabalho, foi desenvolvida uma arquitetura com modos vulnerável e seguro, orientada por princípios de segurança de aplicações (*Application Security*). O ecossistema foi projetado para alternar dinamicamente seu estado por meio de um sinalizador global de segurança no *front-end*, responsável por inserir um cabeçalho personalizado de controle em todas as requisições HTTP enviadas ao servidor.

Esse mecanismo permitiu isolar o comportamento vulnerável para fins didáticos e aplicar as contramedidas técnicas de forma controlada, viabilizando a comparação direta entre os dois estados da aplicação.

[7] Disponível em: https://github.com/s0md3v/XSStrike

4.2.3.1 Mecanismos de Defesa no *Back-end*

No lado do servidor, a arquitetura de segurança foi estruturada em múltiplas camadas de proteção para reduzir a superfície de ataque e fortalecer os mecanismos de defesa da aplicação. As contramedidas implementadas atuam de forma complementar, abrangendo desde o tratamento de dados recebidos até a proteção dos mecanismos de autenticação e da execução de conteúdo no navegador.

A primeira camada consistiu na implementação de uma Política de Segurança de Conteúdo (*Content Security Policy – CSP*), ativada exclusivamente no modo seguro por meio de um *middleware* de segurança. As diretivas configuradas restringem o carregamento de recursos a origens previamente autorizadas e bloqueiam a execução de *scripts inline* e de conteúdos externos não confiáveis. Dessa forma, mesmo que uma carga maliciosa alcance a interface da aplicação, sua execução tende a ser impedida pelas políticas aplicadas pelo navegador.

A segunda camada concentrou-se na sanitização das entradas de dados. Para isso, foi utilizada uma biblioteca especializada no tratamento de *strings*, responsável por filtrar e remover elementos HTML potencialmente perigosos antes que os dados fossem processados ou armazenados. Esse mecanismo foi integrado às rotas de inserção de comentários e aos pontos de demonstração de XSS Refletido, garantindo que conteúdos maliciosos fossem neutralizados e interpretados apenas como texto comum.

Complementando essas camadas, foi implementada proteção voltada ao gerenciamento de sessões por meio da configuração da *flag HttpOnly* nos *cookies* de autenticação gerados pelo servidor. Quando essa propriedade é habilitada, o navegador impede que o conteúdo do *cookie* seja acessado pela API *document.cookie*, bloqueando qualquer tentativa de leitura ou exfiltração por *scripts JavaScript* executados na página. Dessa forma, mesmo que um *payload* malicioso seja injetado com sucesso na interface, o *token* de sessão do usuário permanece inacessível ao código do lado do cliente, limitando diretamente o impacto de ataques de sequestro de sessão (*session hijacking*) associados ao XSS Armazenado e ao XSS Refletido.

4.2.3.2 Mecanismos de Defesa no *Front-end*

No lado do cliente, as estratégias de mitigação concentraram-se na eliminação de pontos de saída inseguros presentes na árvore do DOM e na adoção de práticas seguras para o gerenciamento do estado da aplicação. Uma das principais medidas implementadas foi a adaptação

do componente responsável pela renderização dos dados, que passou a verificar o estado de segurança da aplicação antes de exibir qualquer conteúdo. Quando o modo seguro é ativado, a manipulação direta de elementos por meio de *innerHTML* é desabilitada, e a inserção de dados passa a ocorrer exclusivamente por métodos seguros de renderização textual. Esse tratamento foi aplicado às funcionalidades de busca, listagem de comentários e finalização de pedidos. Como consequência, os *payloads* maliciosos deixaram de ser interpretados pelo navegador e passaram a ser exibidos apenas como texto, impedindo sua execução e reduzindo a necessidade de mecanismos adicionais de tratamento do DOM.

Além disso, foram adotadas medidas específicas para mitigar vulnerabilidades baseadas na manipulação do DOM. As entradas provenientes diretamente da URL, especialmente aquelas recebidas por meio de parâmetros de consulta, passaram a ser submetidas ao mesmo fluxo de renderização segura utilizado nas demais funcionalidades da interface. Com essa abordagem, tentativas de injeção originadas da manipulação local do contexto do navegador foram neutralizadas, reduzindo significativamente os riscos associados às vulnerabilidades do tipo *DOM-based XSS*.

Com a integração dessas contramedidas às camadas de apresentação e de aplicação, consolidou-se a arquitetura de segurança do ecossistema laboratorial, estabelecendo um ambiente controlado e consistente para a análise comparativa das vulnerabilidades de *Cross-Site Scripting (XSS)*. A combinação entre a modelagem intencional das falhas, a execução orientada de vetores de ataque, a aplicação das estratégias de mitigação e a realização de auditorias automatizadas permitiu validar a robustez técnica da plataforma desenvolvida. Com a infraestrutura e os procedimentos experimentais devidamente implementados e verificados, o ecossistema mostrou-se apto para a etapa de validação pedagógica, fornecendo a base necessária para a coleta e análise dos dados, a avaliação da eficácia das mitigações e a investigação de seu impacto no processo de ensino e aprendizagem, cujos resultados são apresentados e discutidos no capítulo seguinte.

5 RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados obtidos a partir dos ensaios experimentais realizados no ambiente laboratorial de *e-commerce*, bem como a validação pedagógica da plataforma como ferramenta didática. A análise foi estruturada em três etapas: a demonstração prática dos vetores de ataque no ambiente vulnerável, a avaliação da eficácia das contramedidas no ambiente seguro e a mensuração do impacto educacional da plataforma junto aos estudantes de graduação.

5.1 Análise dos Vetores de Ataque (ambiente vulnerável)

Para balizar a coleta desses resultados, a eficácia das contramedidas implementadas foi avaliada por meio de um conjunto de testes comparativos e auditorias automatizadas, conduzidos visando verificar o comportamento da aplicação nos modos vulnerável e seguro. Inicialmente, foram realizados testes empíricos de injeção (*fuzzing* manual), nos quais vetores de ataque conhecidos foram inseridos nos campos de busca e nos formulários de comentários da plataforma. No ambiente vulnerável, observou-se a execução bem-sucedida dos *scripts* maliciosos, confirmando a presença das falhas modeladas. Em contrapartida, no modo seguro, os conteúdos injetados passaram a ser exibidos apenas como texto comum na interface, demonstrando a eficácia dos mecanismos de renderização segura adotados.

Complementarmente, realizou-se uma análise do comportamento dos identificadores de sessão e do contexto de execução da aplicação por meio das ferramentas de desenvolvedor (*DevTools*) do navegador. Utilizando uma funcionalidade interna de simulação, foram monitorados os *cookies* e os cabeçalhos HTTP durante diferentes cenários de interação. Os resultados demonstraram que, com as medidas de proteção habilitadas, tentativas de acesso aos dados de autenticação deixaram de expor os *tokens* de sessão, evidenciando a efetividade dos mecanismos de proteção aplicados.

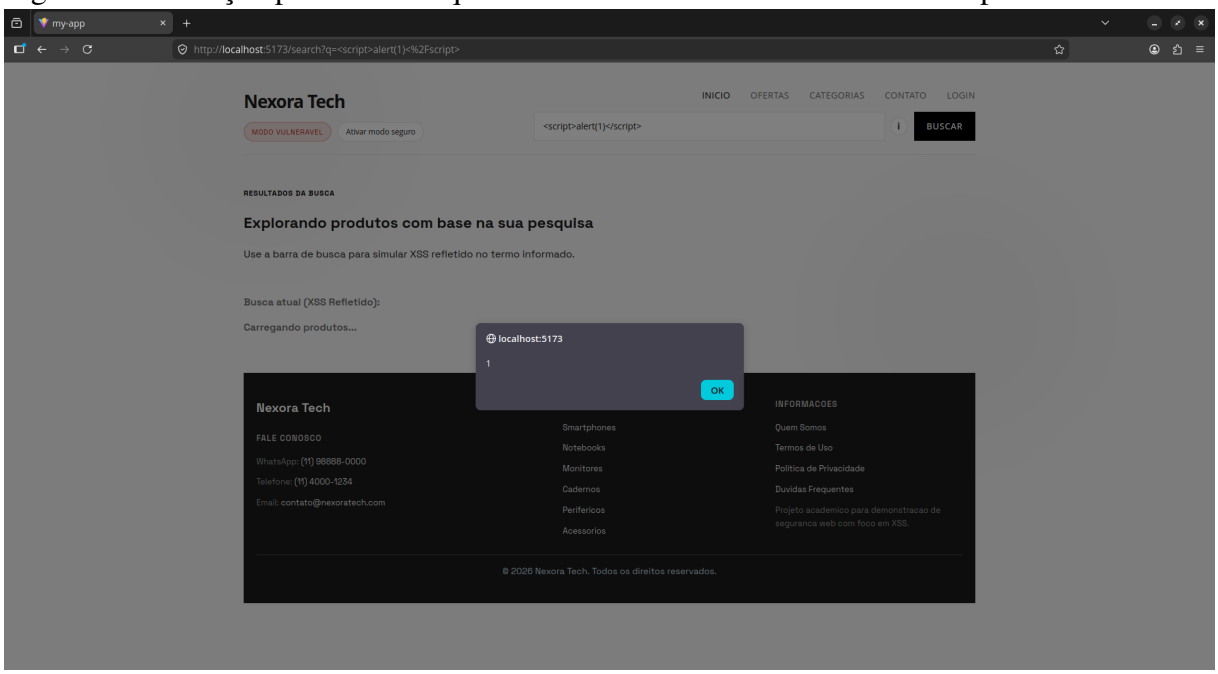
A terceira etapa consistiu em submeter a aplicação a uma auditoria automatizada com a ferramenta XSStrike. A solução foi empregada para executar varreduras de segurança e testes de injeção inteligente (*fuzzing*) nas rotas de busca e nos pontos de entrada de dados da aplicação. A varredura teve por finalidade avaliar a robustez das contramedidas diante de diferentes vetores de ataque e possíveis tentativas de evasão. Os relatórios gerados pela ferramenta confirmaram a neutralização das vulnerabilidades identificadas durante a fase experimental e corroboraram a eficácia dos mecanismos de proteção implementados na arquitetura da aplicação.

5.1.1 Exploração de *Reflected XSS* no Sistema de Busca

No mecanismo de busca de produtos da plataforma, a inserção de um *payload* contendo código executável diretamente no campo de texto foi capturada pelo sistema e refletida imediatamente na resposta da página. A ausência de codificação e filtragem adequadas permitiu que o navegador interpretasse a entrada do usuário como código HTML executável, em vez de renderizá-la como texto simples.

Esse comportamento caracteriza o vetor de *Reflected XSS* (XSS Refletido), no qual o código malicioso é enviado junto à requisição e executado instantaneamente no contexto da sessão do cliente, sem a necessidade de armazenamento prévio. Durante as simulações realizadas no ambiente de desenvolvimento, a injeção do *payload* demonstrou a capacidade de o navegador processar instruções arbitrárias em tempo real, evidenciando a fragilidade das rotas de busca no tratamento e na exibição de parâmetros dinâmicos transmitidos por requisições HTTP

Figura 5 – Execução prática de ataque de XSS Refletido na rota de busca de produtos



Fonte: Autoria própria (2026).

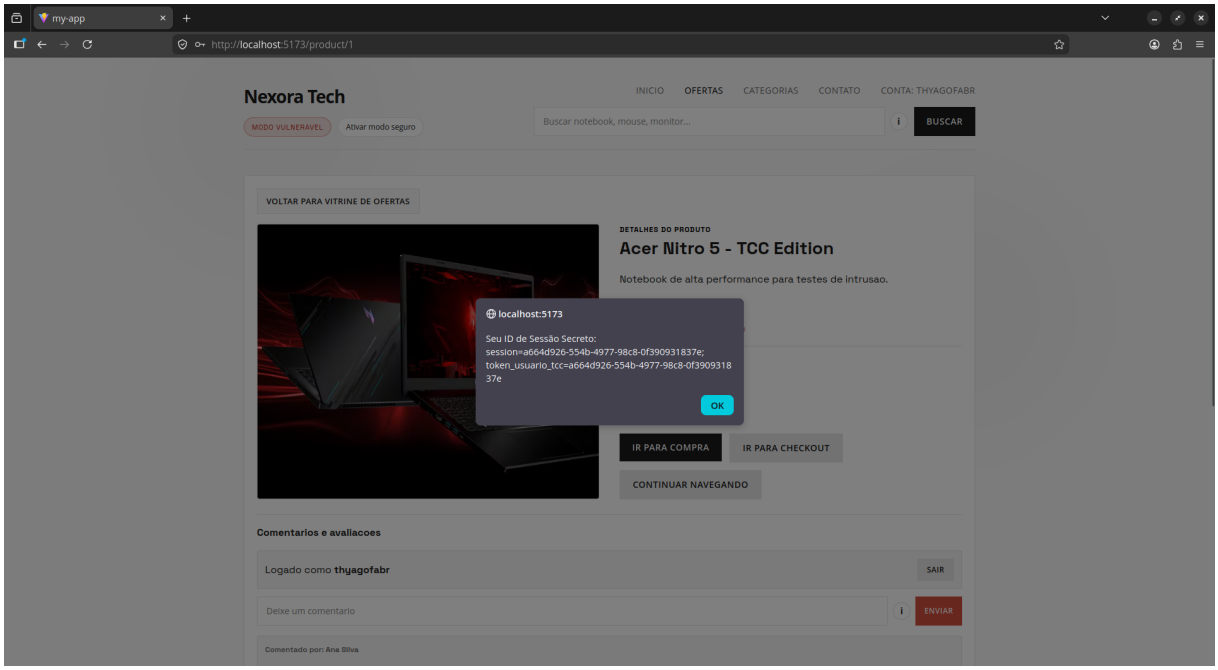
5.1.2 Exploração de *Stored XSS* na Seção de Comentários

Na área destinada às avaliações e comentários dos produtos, o código malicioso inserido por meio do formulário explorou a ausência de filtros e mecanismos de sanitização no servidor,

sendo armazenado integralmente no banco de dados SQLite. Esse cenário caracteriza o vetor *Stored XSS* (XSS Armazenado), no qual o *payload* malicioso permanece persistido no banco de dados da aplicação, conforme descrito nas diretrizes da Owasp (2021).

O impacto crítico dessa vulnerabilidade foi evidenciado durante os ensaios realizados no ambiente laboratorial. Ao acessar a página do produto, o sistema recuperava o conteúdo malicioso armazenado e o inseria diretamente na interface do usuário. Como não havia codificação adequada de caracteres antes da renderização, o navegador interpretava o dado recuperado como código executável. Como consequência, o ataque era disparado automaticamente para qualquer usuário que visitasse a página, demonstrando a possibilidade de exposição de informações sensíveis da sessão e simulando o impacto de um sequestro de sessão.

Figura 6 – Execução do XSS Armazenado com captura de identificador de sessão



Fonte: Autoria própria (2026).

5.1.3 Exploração de *DOM-based XSS* via *Query String*

A validação experimental do terceiro vetor de ataque ocorreu exclusivamente no lado do cliente, sem a necessidade de interação direta ou armazenamento de dados no servidor. Por meio da manipulação da URL da página inicial do *e-commerce*, foi inserido um *payload* contendo código executável diretamente no parâmetro de consulta. A lógica de *front-end* da aplicação utilizou recursos nativos do navegador para capturar esse valor da *query string* e encaminhá-lo

para um ponto de saída inseguro, permitindo que o navegador interpretasse a entrada como código ativo.

Esse cenário caracteriza a vulnerabilidade conhecida como *DOM-based XSS* (XSS Baseado em DOM). Conforme descrito por Klein (2005), esse tipo de ataque diferencia-se dos demais por ocorrer inteiramente na árvore do *DOM (Document Object Model)* durante a execução dos *scripts* no cliente, sem participação direta do servidor no processamento da exploração.

Nos testes realizados no ambiente laboratorial, o impacto desse vetor foi demonstrado por meio de uma técnica de *Defacement* (alteração visual). O *payload* injetado pela URL modificou propriedades de estilo da aplicação, alterando a aparência do *e-commerce* e comprometendo elementos textuais da interface. O experimento evidenciou como dados controlados pelo usuário, quando manipulados de forma insegura no navegador, podem comprometer tanto a identidade visual quanto o comportamento de aplicações web modernas.

Figura 7 – Alteração visual da interface *(Defacement)* gerada por ataque de XSS Baseado em DOM

Fonte: Autoria própria (2026).

5.1.4 Relatório de Auditoria Automatizada Pré-Mitigação

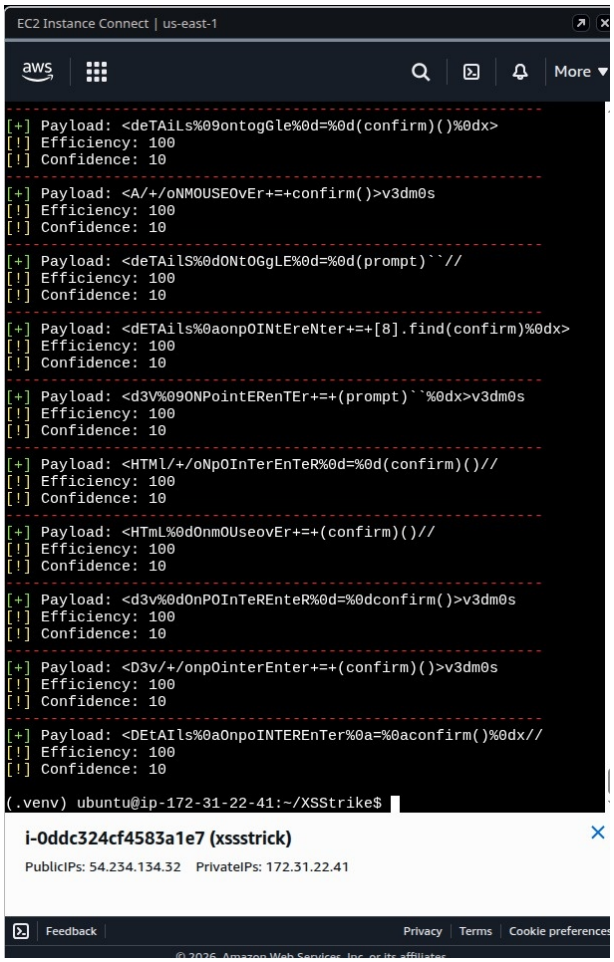
Para complementar as análises manuais e mapear de forma sistemática o nível de exposição do ecossistema, a aplicação em estado inseguro foi submetida a um processo de auditoria

automatizada. Para isso, utilizou-se a ferramenta XSStrike a partir de uma instância *Elastic Compute Cloud* (EC2) da *Amazon Web Services* (AWS), configurada para executar varreduras profundas e testes de injeção inteligente (*fuzzing*) nas rotas de busca, nos parâmetros de URL e nos campos de entrada de dados.

O relatório gerado pela ferramenta confirmou os diagnósticos obtidos nas análises empíricas. A XSStrike identificou com precisão a ausência de mecanismos adequados de codificação e apontou que as rotas avaliadas apresentavam vulnerabilidades críticas associadas a múltiplos vetores de XSS Refletido e *DOM-based XSS*.

Os registros produzidos pela auditoria demonstraram a execução de diferentes categorias de *payloads* capazes de contornar as proteções padrão do navegador. Esses dados confirmaram a necessidade de reestruturação arquitetural do sistema e da adoção de mecanismos defensivos na aplicação.

Figura 8 – Relatório e logs de varredura automatizada com a ferramenta XSStrike em ambiente vulnerável



Fonte: Autoria própria (2026)

5.2 Eficácia das Medidas de Proteção (ambiente seguro)

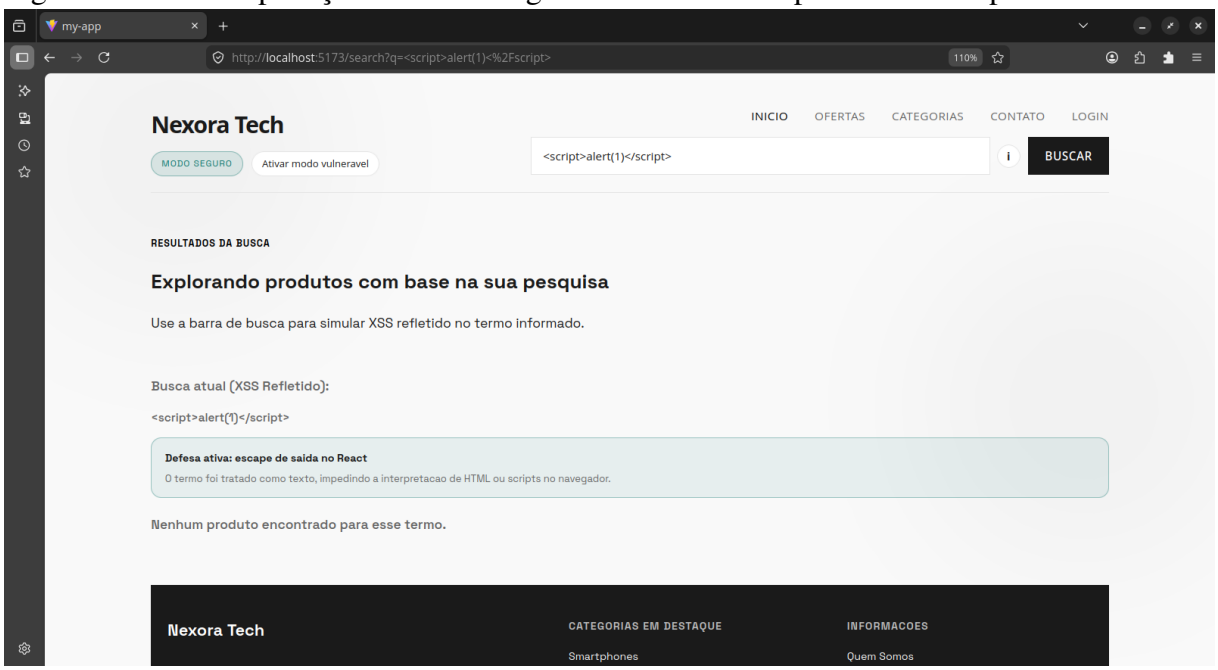
Após a ativação do sinalizador global de segurança na arquitetura do ecossistema, o comportamento da aplicação foi reestruturado por meio da inserção de cabeçalhos de controle e da ativação automática dos mecanismos defensivos. Para avaliar a robustez das barreiras implementadas, a mesma bateria de testes empíricos e auditorias automatizadas realizada na etapa anterior foi repetida no ambiente protegido.

5.2.1 Impacto da Renderização Segura com textContent

No lado do cliente, a primeira linha de defesa avaliada concentrou-se no bloqueio de pontos de saída inseguros presentes na árvore de elementos da página. Quando os mesmos *payloads* maliciosos foram novamente inseridos nos campos de busca de produtos e nos formulários de avaliações, os componentes customizados deixaram de utilizar a propriedade *innerHTML*. Com o modo seguro ativado, o fluxo de dados passou a utilizar métodos seguros de renderização textual, obrigando o navegador a interpretar qualquer conteúdo recebido apenas como texto literal e inofensivo.

Como consequência dessa alteração arquitetural, elementos utilizados em tentativas de ataque perderam a capacidade de induzir o navegador à execução de comandos arbitrários. Os caracteres especiais associados aos vetores de injeção passaram a ser tratados automaticamente e exibidos de forma estática na interface. Esse procedimento neutralizou a execução dos ataques e comprovou a eficácia da renderização segura no tratamento de conteúdos dinâmicos, reduzindo a dependência de mecanismos adicionais de sanitização no navegador para mitigar vulnerabilidades de XSS Refletido e XSS Armazenado.

Figura 9 – Tela da aplicação em modo seguro exibindo os scripts como texto puro



Fonte: Autoria própria (2026)

5.2.2 Validação da Política de Segurança de Conteúdo (CSP)

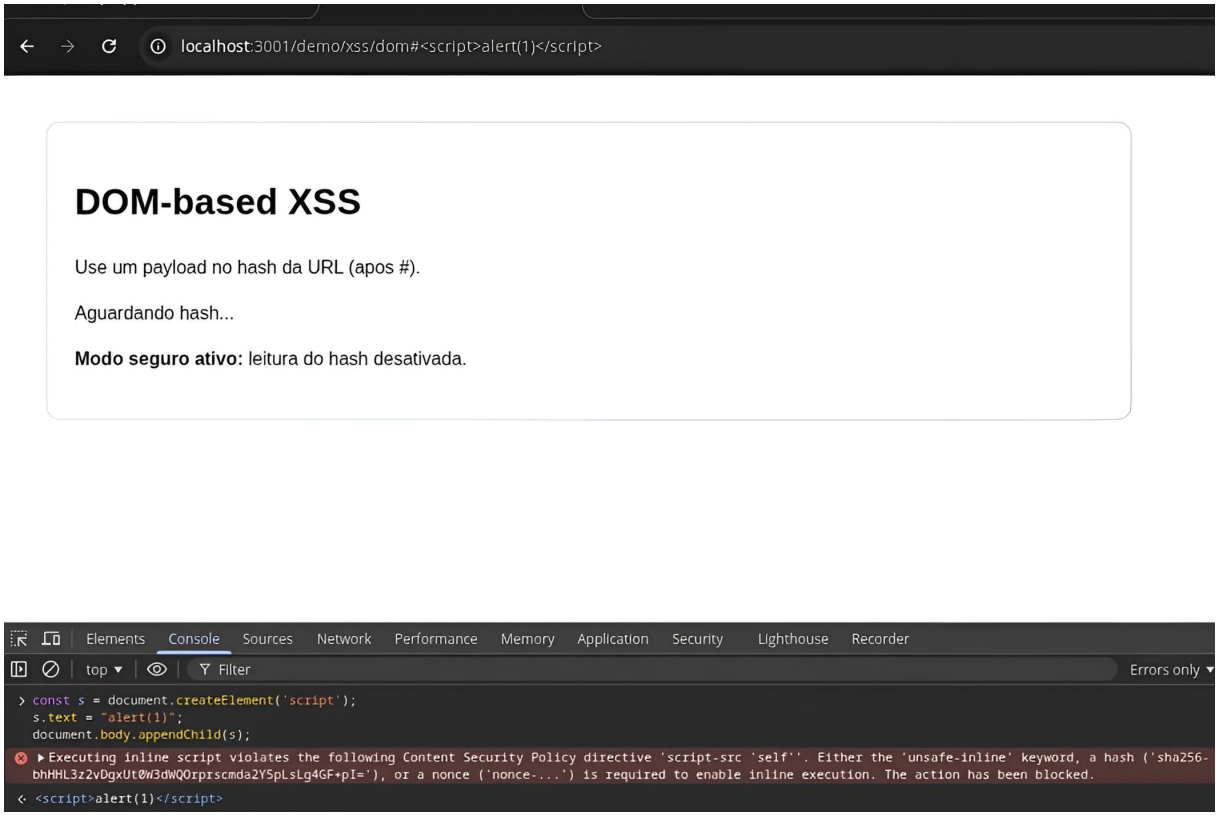
A segunda camada de proteção implementada no ecossistema laboratorial do SecuriShop atuou diretamente na comunicação entre cliente e servidor e no mecanismo de processamento do navegador, por meio da integração do módulo de segurança *Helmet* ao servidor *Express*. A partir da configuração de um *middleware* personalizado, o sistema passou a incluir automaticamente o cabeçalho HTTP *Content Security Policy (CSP)* em todas as respostas enviadas ao cliente. Essa abordagem segue o princípio de defesa em profundidade, estabelecendo regras que limitam as origens autorizadas para o carregamento e a execução de recursos na aplicação.

Para mitigar cenários de evasão frequentemente descritos na literatura, nos quais atacantes tentam executar *scripts* hospedados em servidores externos ou utilizar códigos incorporados diretamente na página, a política de segurança foi configurada de forma restritiva. A diretiva *default-src* foi definida como *self*, permitindo o carregamento de recursos apenas a partir da própria origem da aplicação. Da mesma forma, a diretiva *script-src* também foi definido como *self*, impedindo a execução de *scripts inline* e bloqueando códigos provenientes de domínios ou redes de distribuição de conteúdo externas. Com isso, mesmo que um conteúdo malicioso seja inserido na interface, o navegador é instruído a rejeitar sua execução.

A configuração da política contemplou ainda outras diretivas complementares. A diretiva *style-src* foi definida com os valores *self* e *unsafe-inline*, escolha necessária para garantir o funcionamento adequado dos estilos dinâmicos utilizados pela interface sem comprometer a experiência visual do sistema. Já a diretiva *connect-src* restringiu as requisições assíncronas à própria origem da aplicação e ao domínio especificado na variável de ambiente do cliente (*CLIENT_ORIGIN*). Por sua vez, a diretiva *img-src* limitou o carregamento de imagens ao domínio da aplicação e a conteúdos embutidos por meio do esquema *data:*.

A efetividade dessa camada de proteção foi avaliada por meio da execução dos mesmos vetores de ataque utilizados nos testes anteriores, tanto de forma manual quanto automatizada. Durante os experimentos realizados no modo seguro, o navegador identificou as restrições definidas pela política CSP e bloqueou imediatamente a execução dos códigos não autorizados. Esse comportamento pôde ser observado em tempo real por meio das ferramentas de desenvolvimento do navegador, que registraram mensagens explícitas indicando violações da política de segurança configurada.

Figura 10 – Logs do console do navegador exibindo o bloqueio por violação de CSP



Fonte: Autoria própria (2026)

Os resultados demonstraram que, mesmo nos casos em que conteúdos maliciosos conseguiam alcançar a interface da aplicação, a política CSP impedia sua execução efetiva. Com isso, a proteção implementada atuou como uma barreira adicional contra falhas de programação, reduzindo significativamente os impactos de vulnerabilidades remanescentes e fortalecendo a segurança da aplicação frente a ataques baseados em injeção de código no lado do cliente.

5.2.3 Blindagem de Sessão com *Cookies HttpOnly*

A terceira contramedida validada no ambiente seguro concentrou-se na proteção dos identificadores de sessão (*session tokens*) trafegados entre o cliente e o servidor *Express*. Para isso, o *back-end* foi configurado para aplicar restrições rigorosas por meio da função *configurarCookieSessao*. Quando o sinalizador global de segurança (*isSecureMode*) é ativado, o servidor adiciona automaticamente a propriedade *HttpOnly* aos *cookies* de autenticação gerados pela aplicação. Essa estratégia está alinhada às recomendações do *Application Security Verification Standard (ASVS)* da Owasp (2021), que estabelece a restrição de acesso a credenciais por *scripts* como um requisito fundamental para a segurança no gerenciamento de sessões web.

A efetividade dessa proteção foi validada empiricamente por meio de testes realizados nas ferramentas de desenvolvimento do navegador (*DevTools*), utilizando comandos de inspeção e a instrução *document.cookie*. No modo vulnerável, esse comando permitia visualizar integralmente os *tokens* de sessão armazenados no navegador, possibilitando a simulação de ataques de sequestro de sessão (*session hijacking*). Em contrapartida, quando o modo seguro era ativado, o ambiente de execução *JavaScript* tornava-se incapaz de acessar ou ler o conteúdo dos *cookies*, que permaneciam isolados e inacessíveis aos *scripts* executados no lado do cliente. Os resultados demonstraram que, mesmo diante de uma eventual vulnerabilidade residual de injeção no *front-end*, as credenciais mais sensíveis da sessão permaneceriam protegidas contra exposição direta e tentativas de exfiltração de dados.

Apesar de sua eficácia na mitigação do roubo de credenciais por meio de *scripts*, a literatura especializada aponta limitações importantes relacionadas ao uso isolado da propriedade *HttpOnly*. Embora essa configuração impeça a leitura dos *cookies* por código malicioso, ela não evita que o navegador os envie automaticamente em requisições de rede. Na ausência de mecanismos complementares de proteção, uma vulnerabilidade do tipo *Cross-Site Request Forgery (CSRF)* poderia permitir que um atacante induzisse o navegador da vítima a executar requisições legítimas para o *back-end* utilizando a sessão autenticada do usuário. No contexto do

ecossistema desenvolvido, isso poderia ocorrer, por exemplo, por meio do envio não autorizado de uma requisição HTTP POST para a funcionalidade de cadastro de comentários. Outra limitação relevante está relacionada à transmissão dos *cookies* por canais não criptografados. Caso a comunicação ocorresse por meio de HTTP, os identificadores de sessão poderiam ser interceptados durante o tráfego de rede em ataques do tipo *Man-in-the-Middle (MITM)*.

Para mitigar esses riscos e fortalecer a proteção da aplicação, a arquitetura do *back-end* incorporou mecanismos adicionais de segurança. A propriedade *Secure* foi vinculada dinamicamente à verificação do uso de TLS/HTTPS por meio da condição *req.secure*, garantindo que os *cookies* fossem transmitidos exclusivamente por conexões criptografadas. Além disso, o atributo *SameSite* foi configurado como *Strict*, restringindo o envio automático de *cookies* em requisições originadas de outros domínios. A combinação dessas medidas reduziu significativamente a superfície de ataque associada aos vetores de CSRF e MITM, proporcionando uma camada adicional de proteção aos mecanismos de autenticação implementados no ambiente seguro.

5.2.4 Relatório de Auditoria Automatizada Pós-Mitigação

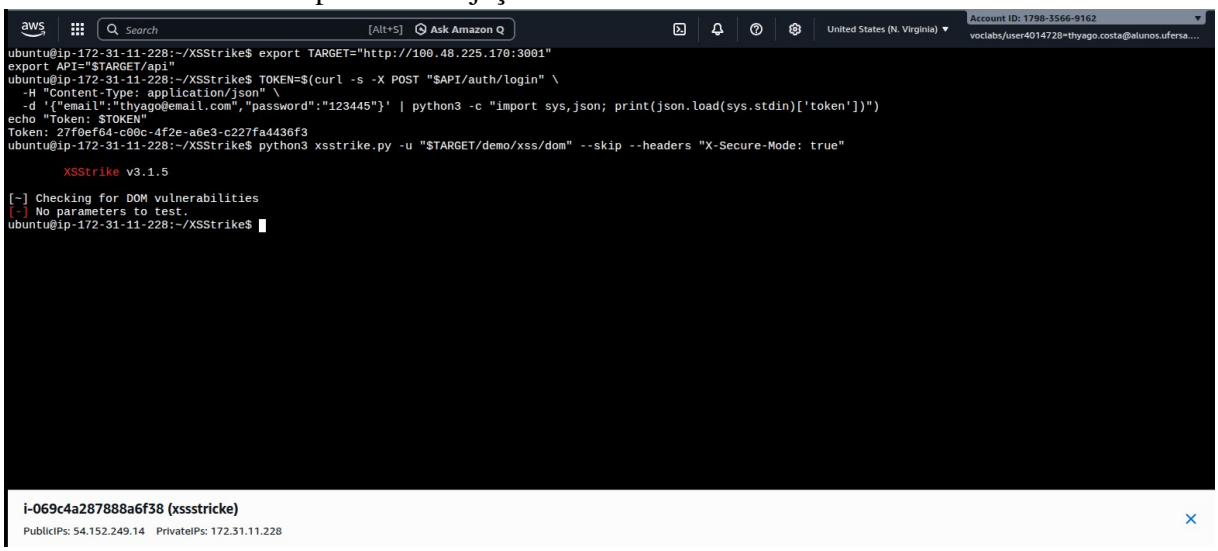
Com o objetivo de validar tecnicamente a arquitetura defensiva implementada e verificar a eliminação dos vetores de injeção previamente identificados, a aplicação *SecuriShop* foi submetida a uma nova auditoria automatizada em ambiente protegido. Para isso, utilizou-se a ferramenta *XSStrike* executada a partir de uma instância *Elastic Compute Cloud* (EC2) da *Amazon Web Services* AWS conectada ao servidor configurado em modo seguro. O protocolo de *fuzzing* inteligente foi reaplicado sobre as mesmas rotas e parâmetros analisados durante a etapa inicial dos testes, com foco nos parâmetros q, presente na rota de busca, e *payload*, utilizado na rota de reflexão.

Os resultados obtidos após a implementação das contramedidas confirmaram a efetividade das barreiras de segurança incorporadas ao ecossistema de *e-commerce*. Durante a análise das rotas protegidas, a ferramenta retornou o diagnóstico "*No parameters to test*" (Nenhum parâmetro testável). No contexto de funcionamento do *XSStrike*, essa resposta indica que o scanner não conseguiu identificar pontos de entrada suscetíveis à exploração ou reflexos dinâmicos capazes de processar cargas maliciosas.

Esse comportamento está diretamente relacionado às medidas de proteção implementadas na aplicação. A sanitização rigorosa dos dados realizada no *back-end* por meio da biblioteca xss, aliada às restrições impostas pela Política de Segurança de Conteúdo (*Content Security Policy* —

CSP), eliminou os pontos de saída inseguros anteriormente exploráveis. Como consequência, os dados submetidos passaram a ser tratados exclusivamente como conteúdo textual, impedindo que as cargas de teste fossem interpretadas como código executável pelo navegador ou refletidas de forma insegura pelas respostas da API.

Figura 11 – Relatório de conformidade da ferramenta XSStrike em ambiente seguro, indicando ausência de pontos de injeção testáveis



Fonte: Autoria própria (2026)

Dessa forma, o mecanismo de análise do *XSStrike* tornou-se incapaz de encontrar caminhos de exploração, reflexos executáveis ou parâmetros vulneráveis, evidenciando a eficácia das estratégias de mitigação adotadas. Quando analisado em conjunto com os testes empíricos realizados manualmente, esse resultado reforça que as contramedidas implementadas foram capazes de neutralizar com sucesso os vetores de *Cross-Site Scripting (XSS)* modelados ao longo do experimento, demonstrando a robustez da arquitetura de segurança proposta para o ambiente SecuriShop.

5.3 Avaliação do Impacto Didático e Validação Empírica

Concluída a validação dos aspectos relacionados à infraestrutura e aos mecanismos de segurança da aplicação, o ecossistema laboratorial foi submetido a uma avaliação pedagógica a fim de mensurar sua eficácia como ferramenta de apoio ao ensino de segurança de software. A amostra foi composta por estudantes de graduação que interagiram com o ambiente de *e-*

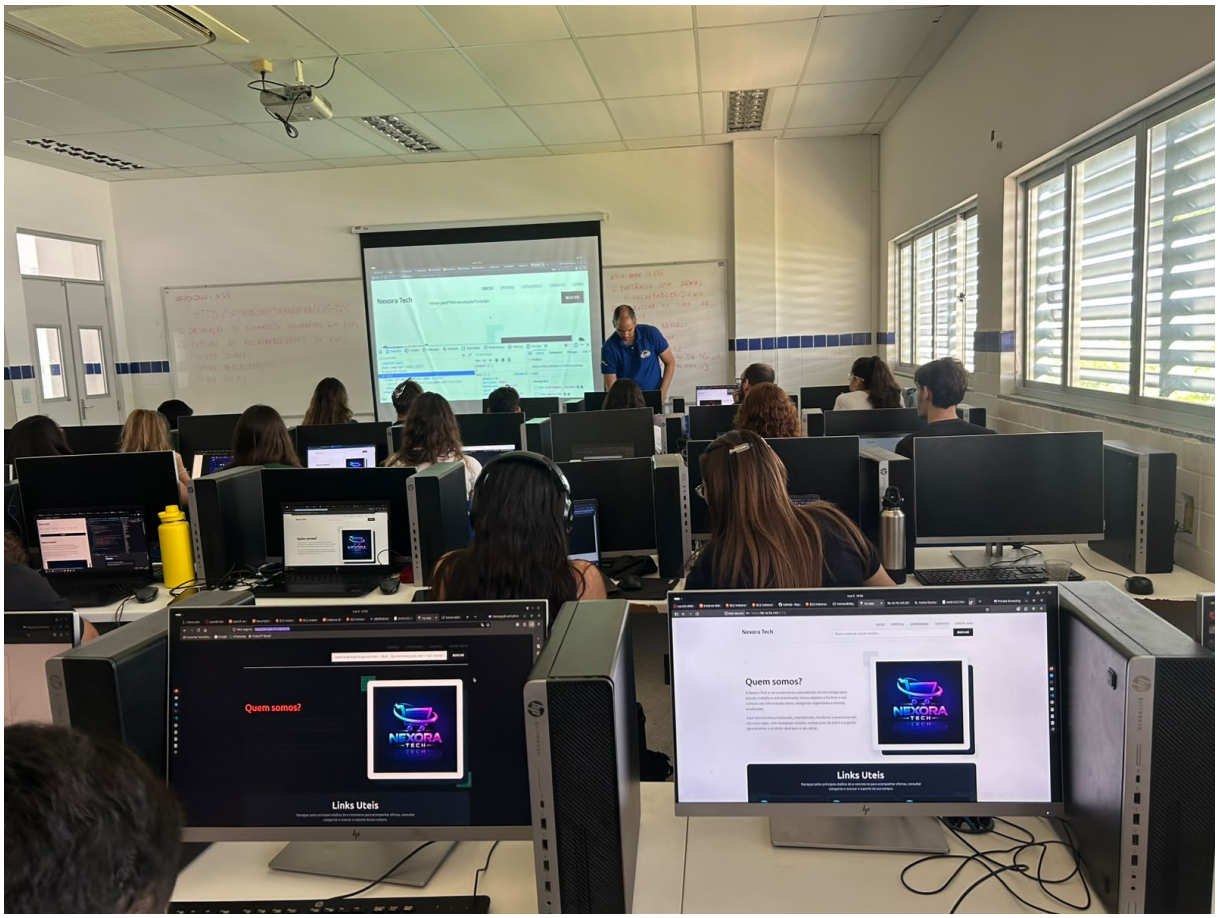
commerce em seu estado vulnerável, tendo a oportunidade de observar a execução de injeções de *scripts* nas rotas da aplicação antes da implementação dos mecanismos defensivos.

Essa estratégia de validação empírica busca responder aos desafios apontados por Neto *et al.* (2024), que evidenciam a necessidade de fortalecer a educação em desenvolvimento seguro durante a formação acadêmica, a fim de atender às demandas da indústria de tecnologia no Brasil. Ao permitir que os estudantes explorem vulnerabilidades em um ambiente controlado, a plataforma atua como suporte pedagógico para promover a conscientização sobre riscos cibernéticos presentes no desenvolvimento de software.

A amostra da pesquisa foi composta por 38 estudantes do curso de Tecnologia da Informação da Universidade Federal Rural do Semi-Árido (UFERSA), campus Pau dos Ferros, matriculados no 6º período. A atividade foi realizada no dia 8 de maio de 2026, ocasião em que 44 estudantes estavam presentes em sala de aula. Desses, 38 aceitaram participar voluntariamente da pesquisa, correspondendo a uma taxa de adesão de aproximadamente 86,4% da turma. A coleta dos dados ocorreu de forma totalmente anônima, sem qualquer identificação individual dos participantes, assegurando o sigilo e a confidencialidade das informações fornecidas. Os dados utilizados nesta análise foram coletados por meio de um questionário estruturado, aplicado após a realização das atividades laboratoriais, totalizando 38 respostas válidas. O instrumento de pesquisa encontra-se documentado no Apêndice A e investigou aspectos relacionados ao ganho de conhecimento dos participantes, à clareza da interface desenvolvida e à capacidade da plataforma de estimular o aprendizado autônomo sobre vulnerabilidades de *Cross-Site Scripting (XSS)*.

Os resultados estatísticos obtidos a partir desse levantamento empírico fundamentam as discussões apresentadas a seguir, relacionando os dados quantitativos às premissas pedagógicas que orientaram o desenvolvimento do projeto.

Figura 12 – Aplicação prática do ecossistema laboratorial didático em ambiente de sala de aula na UFERSA



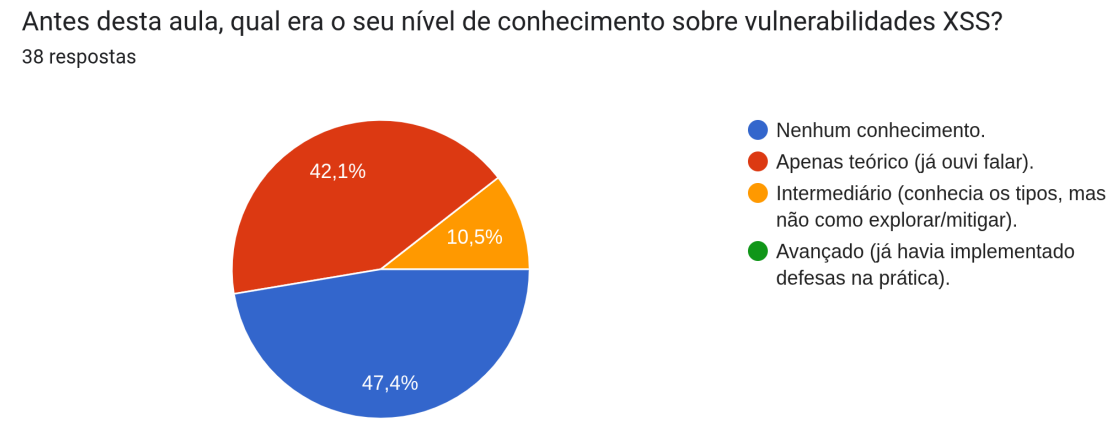
Fonte: Arquivo Pessoal (2026)

5.3.1 Diagnóstico do Nível de Conhecimento Prévio e Percepção de Ataques Práticos

O primeiro indicador obtido a partir do levantamento de dados teve como objetivo identificar o perfil técnico e o nível de familiaridade dos 38 participantes com vulnerabilidades web antes da intervenção laboratorial. Conforme apresentado na Figura 13, os resultados demonstraram que a maior parte da amostra possuía pouco ou nenhum contato prático com vulnerabilidades de *Cross-Site Scripting (XSS)*.

Quando questionados sobre seu domínio teórico e prático em relação a esse tipo de falha, 47,4% dos estudantes afirmaram não possuir conhecimento sobre o tema, enquanto 42,1% declararam ter apenas uma compreensão teórica, limitada a conceitos apresentados de forma passiva. Apenas 10,5% dos participantes indicaram possuir um nível intermediário de familiaridade com o vetor de ataque.

Figura 13 – Percepção dos estudantes quanto ao nível de conhecimento sobre XSS antes da prática laboratorial

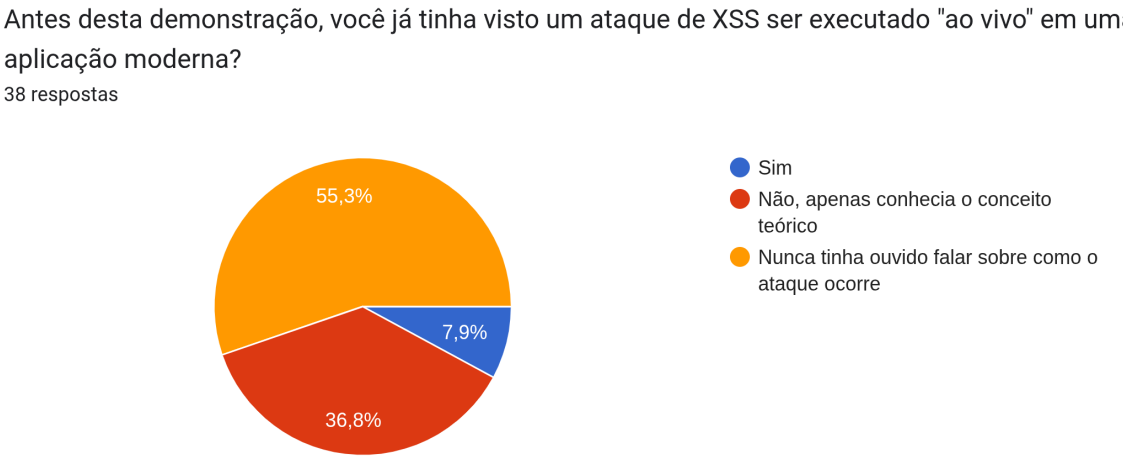


Fonte: Autoria própria (2026)

Esse cenário de distanciamento prático torna-se ainda mais evidente quando analisada a experiência prévia dos estudantes em relação à execução de ataques em ambientes reais de software. Os dados apresentados na Figura 14 mostram que 55,3% dos participantes nunca haviam tido contato com a demonstração prática de um ataque de XSS executado em tempo real em um sistema moderno. Além disso, 36,8% afirmaram conhecer o conceito apenas de forma teórica, por meio de descrições genéricas, sem qualquer experiência em ambientes funcionais. Apenas 7,9% dos estudantes já haviam presenciado uma demonstração prática semelhante.

Esse panorama reforça as discussões apresentadas por Neto *et al.* (2024) sobre o distanciamento existente entre as abordagens tradicionais do ensino superior e as competências práticas de segurança exigidas pelo mercado de tecnologia. Nesse contexto, o ecossistema de *e-commerce* desenvolvido mostrou-se relevante ao proporcionar aos estudantes um primeiro contato prático com a manipulação e a observação direta de vulnerabilidades de injeção em aplicações web reais.

Figura 14 – Experiência prévia dos participantes com a execução de ataques XSS em tempo real.



Fonte: Autoria própria (2026).

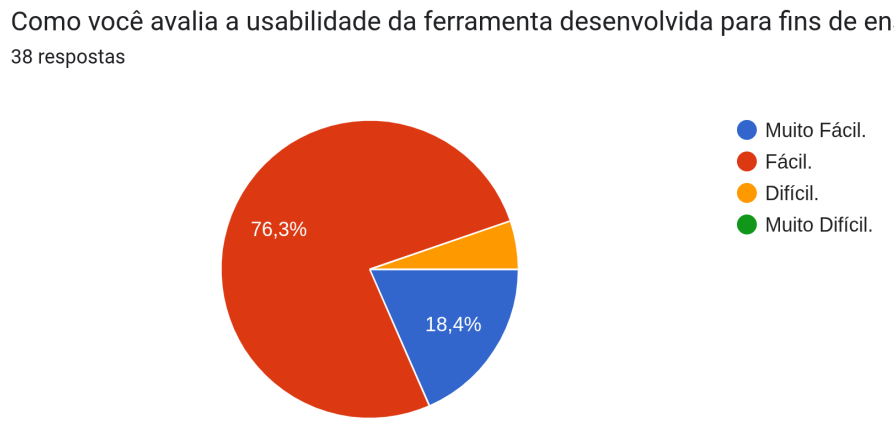
5.3.2 Avaliação da Usabilidade e Interface do Ecossistema

A aceitação de uma ferramenta educacional voltada à engenharia de software está diretamente associada à sua facilidade de uso, evitando que a complexidade da interface gráfica se transforme em uma barreira adicional ao processo de aprendizagem. Ao analisar a percepção dos 38 participantes sobre a usabilidade do SecuriShop desenvolvido, os resultados apontaram ampla aceitação da interface, conforme detalhado a seguir.

Conforme apresentado na Figura 15, a maioria dos estudantes avaliou positivamente a experiência de interação com a plataforma. Do total de participantes, 76,3% classificaram a usabilidade do sistema como “Fácil”, enquanto 18,4% a definiram como “Muito Fácil”. Somados, esses indicadores representam 94,7% de avaliações favoráveis, evidenciando que o *layout*, a organização dos elementos do *e-commerce* e o fluxo de navegação foram suficientemente intuitivos para permitir que os estudantes concentrassem sua atenção na compreensão dos conceitos de segurança e na observação das injeções de scripts em tempo real.

Apenas 5,3% dos respondentes consideraram a usabilidade do sistema “Difícil”, enquanto nenhum participante a classificou como “Muito Difícil”. Esses resultados reforçam a eficiência do *design* da interface e demonstram que o ambiente cumpriu adequadamente seu papel como suporte instrucional, reduzindo barreiras técnicas relacionadas à configuração e ao uso da plataforma pelos estudantes.

Figura 15 – Percepção dos estudantes quanto à usabilidade da ferramenta desenvolvida para fins de ensino



Fonte: Autoria própria (2026)

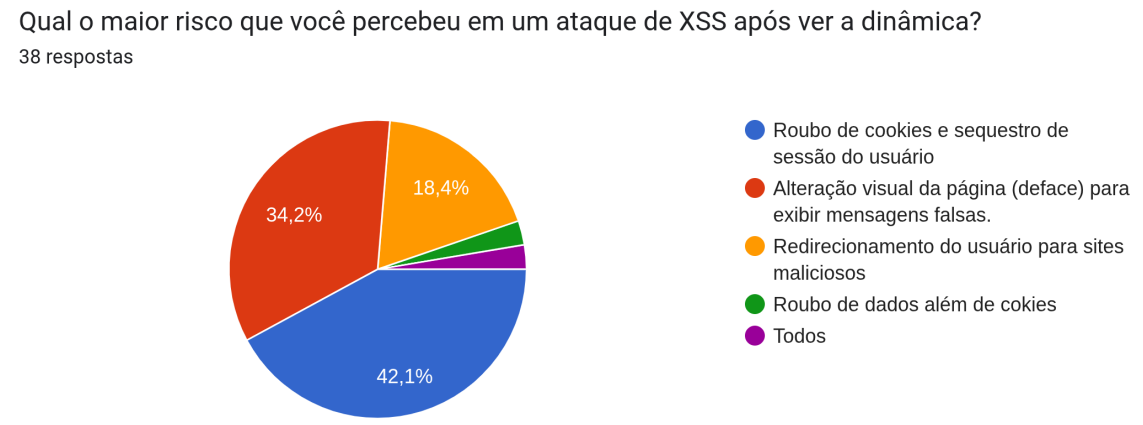
5.3.3 Eficácia da Exploração Prática na Percepção de Segurança

O último aspecto avaliado buscou mensurar a capacidade da dinâmica laboratorial de conscientizar os estudantes sobre a gravidade e os impactos reais das vulnerabilidades de *Cross-Site Scripting (XSS)*. A visualização dos ataques sendo executados em tempo real proporcionou aos participantes uma compreensão prática do potencial nocivo dessas falhas, superando a percepção puramente teórica que frequentemente minimiza as consequências do XSS no desenvolvimento de software.

Conforme apresentado na Figura 16, ao serem questionados sobre o principal risco percebido após a demonstração, a maioria dos estudantes concentrou suas respostas em impactos críticos relacionados à privacidade e à segurança das sessões dos usuários. O "Roubo de *cookies* e sequestro de sessão do usuário" foi apontado por 42,1% dos participantes como a ameaça mais preocupante, evidenciando que os estudantes compreenderam como a exposição de dados locais no navegador pode comprometer credenciais ativas.

A "Alteração visual da página (*Defacement*) para exibição de mensagens falsas" apareceu como o segundo risco mais citado, representando 34,2% das respostas e refletindo diretamente os cenários explorados durante os ensaios práticos no laboratório. Já o "Redirecionamento do usuário para sites maliciosos" foi mencionado por 18,4% dos participantes. As opções residuais "Roubo de dados além de *cookies*" e "Todos" corresponderam a 2,6% das respostas cada.

Figura 16 – Percepção dos estudantes quanto ao maior risco associado a um ataque de XSS após a dinâmica prática

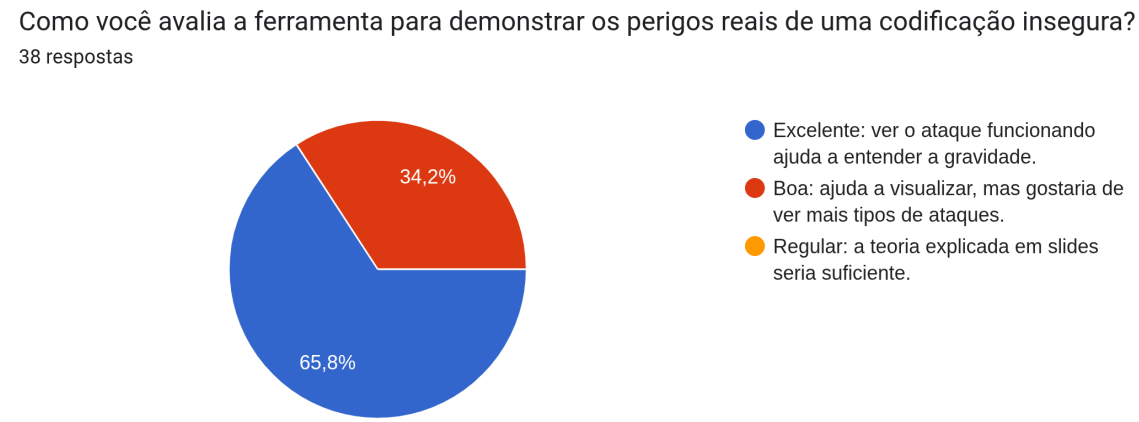


Fonte: Autoria própria (2026)

Essa maturidade na percepção dos riscos também foi refletida na avaliação dos estudantes sobre a capacidade pedagógica da ferramenta em demonstrar os impactos de uma codificação insegura. Os dados apresentados na Figura 17 mostram que 65,8% dos participantes classificaram a plataforma como “Excelente”, destacando que visualizar o ataque em funcionamento contribuiu diretamente para a compreensão da gravidade do problema em cenários reais. Os 34,2% restantes avaliaram a ferramenta como “Boa”, afirmando que o ambiente auxiliou de maneira significativa na visualização prática das vulnerabilidades.

Esse elevado nível de aprovação reforça as conclusões apresentadas por Neto *et al.* (2024) sobre a relevância de abordagens empíricas no processo de letramento digital e formação em segurança de software. Ao presenciarem a exploração controlada da aplicação, os estudantes deixaram de ter apenas uma compreensão conceitual do problema e passaram a assimilar, de forma mais concreta, a importância de desenvolver sistemas orientados por práticas e diretrizes de segurança.

Figura 17 – Avaliação da ferramenta para demonstrar os perigos reais de uma codificação insegura



Fonte: Autoria própria (2026)

5.3.4 Desmistificação de Segurança em *Frameworks* Modernos e Absorção de Conceitos

Um dos principais objetivos do ecossistema laboratorial era verificar se a experiência prática seria capaz de desmistificar a crença de que *frameworks* modernos são, por si só, totalmente seguros contra falhas de segurança. Conforme apresentado na Figura 18, ao serem questionados se a utilização da ferramenta contribuiu para compreender que tecnologias como *React* e *Next.js* não são completamente seguras de forma isolada, a maioria dos participantes demonstrou uma mudança significativa de percepção.

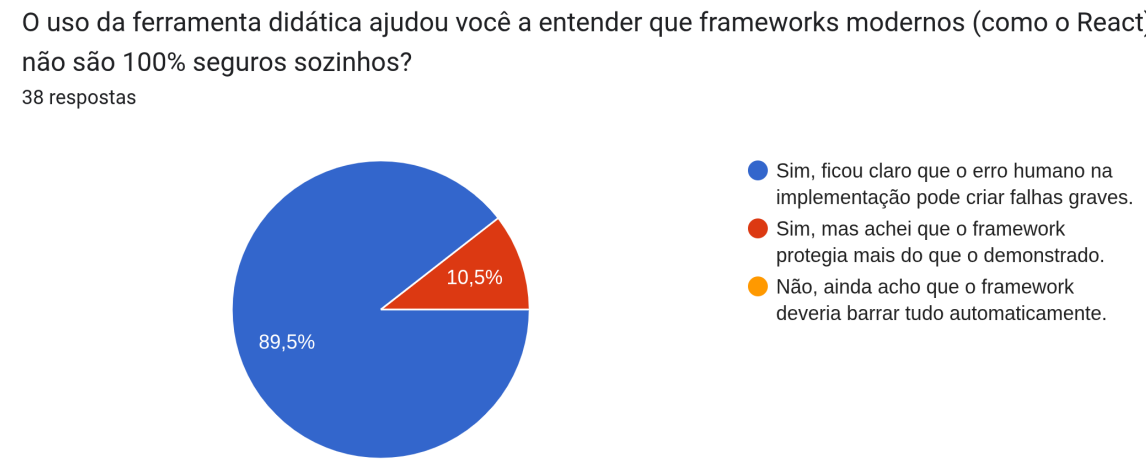
Os resultados indicam que 89,5% dos estudantes responderam de forma categórica que sim, reconhecendo que falhas humanas na implementação de rotas e componentes podem introduzir vulnerabilidades graves na aplicação. Os 10,5% restantes também responderam positivamente, embora tenham destacado que imaginavam que os *frameworks* oferecessem um nível de proteção padrão maior do que o observado durante os testes.

Esse elevado nível de conscientização sobre a influência do fator humano torna-se ainda mais relevante quando analisado à luz do cenário tecnológico atual, marcado pela crescente adoção da Inteligência Artificial(IA) no desenvolvimento de software. Esse contexto reforça as diretrizes da OWASP Foundation (2025) relacionadas à vulnerabilidade *LLM05: Improper Output Handling*, que alerta para os riscos associados à utilização indiscriminada de códigos gerados por modelos de linguagem. Quando sugestões produzidas por sistemas de IA são incorporadas

sem a devida análise crítica e sem o suporte de conhecimentos sólidos em segurança de software, vulnerabilidades graves, como o *Cross-Site Scripting (XSS)*, podem ser introduzidas diretamente nas aplicações web.

Essa constatação reforça a compreensão de que *frameworks* modernos, bibliotecas especializadas e assistentes baseados em Inteligência Artificial (IA) não substituem a responsabilidade do desenvolvedor na construção de sistemas seguros. Independentemente das ferramentas utilizadas, a decisão final sobre a qualidade e a segurança do código permanece sob responsabilidade do programador, que deve adotar uma postura crítica e fundamentada para garantir a construção de aplicações resilientes e confiáveis.

Figura 18 – Percepção dos participantes quanto à segurança nativa de frameworks modernos após a dinâmica



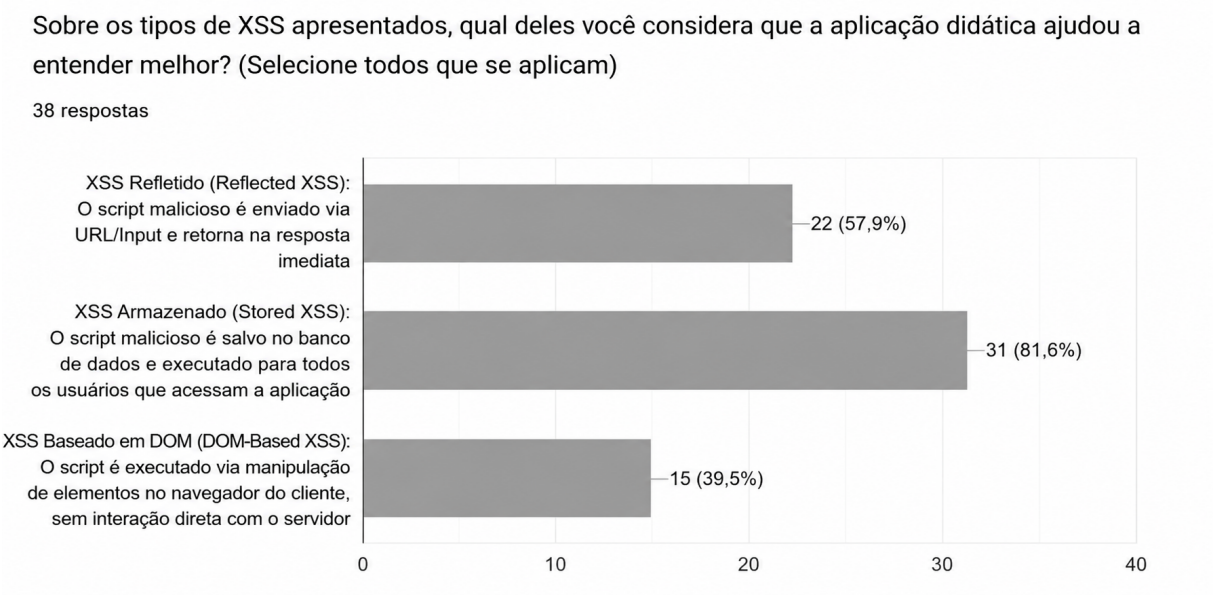
Fonte: Autoria própria (2026)

Essa quebra de paradigma também permitiu analisar o nível de assimilação dos estudantes em relação às diferentes categorias de injeção apresentadas durante os testes realizados na aplicação de *e-commerce*. Conforme os dados consolidados na Figura 19, o XSS Armazenado destacou-se como o vetor de mais fácil compreensão e identificação pelos participantes, seguido pelo XSS Refletido.

O forte impacto visual causado pela persistência do *script* malicioso no banco de dados e pela sua execução em múltiplos acessos posteriores contribuiu para que a lógica de funcionamento desse tipo de ataque fosse assimilada de maneira mais imediata pelos estudantes. Em contrapartida, o XSS Baseado em DOM apresentou um índice menor de compreensão

inicial, refletindo sua maior complexidade técnica, especialmente por envolver a manipulação de elementos e fluxos de dados processados exclusivamente no lado do cliente.

Figura 19 – Mapeamento dos tipos de XSS compreendidos por meio da aplicação didática



Fonte: Autoria própria (2026)

5.3.5 Validação Metodológica e Engajamento para Aprendizado Futuro

Para consolidar a validação empírica da plataforma sob uma perspectiva estritamente pedagógica, o questionário também investigou o valor da abordagem baseada em laboratório prático em comparação aos métodos tradicionais de ensino expositivo. Os dados apresentados na Figura 20 demonstram um cenário de aprovação unânime em relação à estratégia educacional adotada.

Do total de participantes, 71,1% afirmaram de forma categórica que o ambiente prático facilitou significativamente a fixação do conteúdo, destacando que acompanhar a execução do ataque “ao vivo” contribuiu de maneira mais efetiva para o aprendizado. Os 28,9% restantes avaliaram que o uso do sistema de *e-commerce* vulnerável atuou de forma excelente como complemento aos fundamentos teóricos apresentados em sala de aula.

Além disso, nenhum participante indicou que a dinâmica foi indiferente ou demonstrou preferência por uma abordagem exclusivamente teórica, resultado que reforça a relevância de ecossistemas empíricos no processo de ensino e retenção de conceitos relacionados à segurança de software.

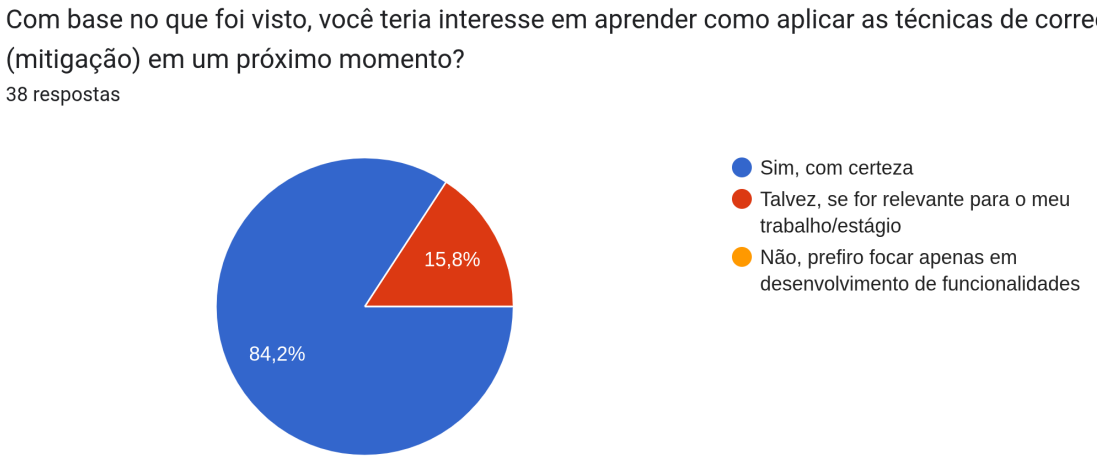
Figura 20 – Percepção dos estudantes sobre o impacto do ambiente prático na fixação de conteúdo versus aula tradicional

Fonte: Autoria própria (2026)

O resultado de maior relevância pedagógica do ecossistema laboratorial refletiu-se na capacidade de despertar o interesse dos estudantes pela continuidade dos estudos em engenharia de software seguro. Conforme apresentado na Figura 21, a ampla maioria dos participantes, correspondente a 84,2% da amostra, declarou ter total interesse em aprender, futuramente, como aplicar técnicas de correção e mitigação de vulnerabilidades. Os 15,8% restantes também demonstraram interesse, embora condicionado à relevância dessas práticas em seus contextos de trabalho ou estágio.

O fato de nenhum estudante manifestar desinteresse ou indicar preferência exclusiva pelo desenvolvimento tradicional de funcionalidades reforça, de forma significativa, os resultados alcançados pelo projeto. A exploração guiada de falhas reais não apenas contribuiu para a compreensão dos riscos associados às vulnerabilidades de injeção, mas também atuou como um importante estímulo para que futuros profissionais busquem capacitação em desenvolvimento seguro, criação de aplicações resilientes e implementação de mecanismos de defesa eficientes. Esse resultado está alinhado às perspectivas educacionais defendidas por Neto *et al.* (2024).

Figura 21 – Grau de interesse dos estudantes em aprender técnicas de correção e mitigação pós-dinâmica



Fonte: Autoria própria (2026)

5.4 Discussão com os trabalhos Relacionados

Diversas iniciativas acadêmicas e ferramentas *open source* têm sido utilizadas para apoiar o ensino prático de segurança em aplicações web por meio de ambientes controlados de experimentação. Entre elas, uma que se destaca é o *OWASP Juice Shop*, uma aplicação de *e-commerce* intencionalmente vulnerável amplamente empregada em treinamentos de segurança ofensiva e defensiva. No cenário nacional, Rosa *et al.* (2024) demonstraram a eficácia do uso dessa plataforma como ambiente de testes, evidenciando que ferramentas de auditoria automatizada alcançam diferentes níveis de cobertura ao analisar os endpoints e as rotas de API que compõem os fluxos de uma aplicação comercial. Esse resultado dialoga diretamente com os procedimentos adotados nesta pesquisa, que utilizou o *scanner* automatizado *XSStrike* para avaliar a robustez das proteções implementadas no ambiente desenvolvido.

No contexto da formação em desenvolvimento seguro, Neto *et al.* (2024) apontam que uma parcela significativa das equipes de software ainda apresenta dificuldades para identificar e corrigir vulnerabilidades em seus próprios sistemas. Esse cenário reforça a importância de iniciativas que aproximem os estudantes de situações práticas de segurança ainda durante a graduação, contribuindo para o desenvolvimento de competências frequentemente exigidas pelo mercado.

Sob a perspectiva educacional, os resultados obtidos com os 38 estudantes participantes deste estudo apresentam convergência com os achados de Bertoglio *et al.* (2022), no âmbito do projeto *Weasels*. Embora a proposta do *Weasels* esteja baseada em atividades progressivas realizadas em plataformas externas, enquanto este trabalho utiliza uma infraestrutura própria construída sobre *frameworks* modernos, ambas as abordagens evidenciam que a aprendizagem prática favorece maior engajamento e melhor assimilação dos conceitos de segurança. De forma semelhante, Queiroz e Cavalcanti (2024) destacam que a interação direta com cenários de vulnerabilidade e a observação imediata dos impactos das falhas contribuem significativamente para a consolidação do conhecimento técnico.

À luz da literatura analisada, este trabalho diferencia-se das iniciativas semelhantes por algumas características específicas. Primeiramente, concentra-se exclusivamente no estudo das vulnerabilidades de *Cross-Site Scripting (XSS)* em aplicações desenvolvidas com *React*, um dos *frameworks* mais utilizados no desenvolvimento web contemporâneo. Além disso, a plataforma foi projetada com uma arquitetura de operação alternável entre cenários vulneráveis e protegidos, permitindo a comparação direta entre a exploração das falhas e a aplicação das respectivas contramedidas. Por fim, o estudo incorpora uma etapa de validação pedagógica com estudantes de graduação da UFERSA, produzindo evidências quantitativas sobre o impacto da abordagem proposta no processo de ensino e aprendizagem de segurança de software.

6 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho alcançou com êxito o objetivo de projetar, desenvolver e validar empiricamente um ecossistema laboratorial voltado ao ensino prático de vulnerabilidades de *Cross Site Scripting (XSS)* no contexto de *frameworks* modernos de desenvolvimento web. A construção de uma aplicação simulada de comércio eletrônico (*e-commerce*), baseada em tecnologias como *React* e *Tailwind CSS*, demonstrou elevado potencial pedagógico. O ambiente desenvolvido contribuiu para suprir uma lacuna recorrente na formação acadêmica de novos engenheiros de software: a ausência de experiências práticas com segurança de aplicações.

A investigação inicial evidenciou que, embora *frameworks* modernos disponham de mecanismos nativos de proteção contra injeções de código, como o escapamento automático de *strings* presente no ecossistema *React*, a dependência exclusiva dessas barreiras automatizadas pode gerar uma falsa percepção de segurança entre desenvolvedores. Por meio da implementação controlada de rotas vulneráveis, foi possível demonstrar, na prática, que falhas humanas na lógica da aplicação e o tratamento inadequado de dados no lado do cliente são suficientes para subverter as defesas do *framework* e expor os usuários a riscos críticos de segurança. A validação empírica conduzida com 38 estudantes de graduação forneceu resultados estatísticos consistentes que comprovam o impacto didático positivo da ferramenta desenvolvida. O diagnóstico inicial revelou um cenário preocupante, no qual a maior parte dos participantes nunca havia presenciado a execução prática de um ataque real ou possuía apenas conhecimentos abstratos sobre segurança de aplicações web. A dinâmica laboratorial em tempo real contribuiu para superar essa limitação conceitual, promovendo uma evolução significativa na percepção técnica dos estudantes quanto aos riscos de segurança, resultado alinhado às recomendações de Neto *et al.* (2024) sobre a formação prática em desenvolvimento seguro.

Além disso, os elevados índices de aprovação relacionados à usabilidade da interface gráfica, com 94,7% de avaliações favoráveis, demonstraram que o *design* centrado no usuário cumpriu adequadamente seu papel como suporte instrucional. A interface intuitiva reduziu barreiras operacionais secundárias e permitiu que os participantes direcionassem sua atenção ao aprendizado técnico e à compreensão dos impactos das vulnerabilidades exploradas.

A observação prática da exploração das falhas correlaciona-se com o interesse da maioria dos estudantes em aprender técnicas de correção e mitigação de vulnerabilidades. Esse resultado indica que a experimentação prática em ambientes controlados atua como um suporte complementar para a fixação de conceitos quando comparada às abordagens exclusivamente expositivas,

auxiliando no processo de ensino de segurança de software.

6.1 Limitações

Apesar dos resultados positivos obtidos, este trabalho apresenta algumas limitações que devem ser consideradas na interpretação dos dados. A amostra composta por 38 estudantes, embora suficiente para uma validação inicial da proposta, está restrita a um único contexto institucional, o que reduz a possibilidade de generalização dos resultados para diferentes perfis acadêmicos e cenários educacionais.

Além disso, a ausência de um grupo de controle formado por estudantes submetidos exclusivamente a métodos tradicionais de ensino expositivo impossibilita uma comparação estatística direta entre as abordagens pedagógicas utilizadas. Assim, os resultados obtidos concentram-se principalmente na percepção e na experiência prática dos participantes durante a dinâmica laboratorial.

Outra limitação relevante refere-se à necessidade de desenvolvimento de um componente customizado para contornar as proteções nativas do *React*. Embora essa decisão tenha sido fundamentada em cenários reais de uso inadequado documentados pela própria biblioteca, a simplificação controlada do ambiente experimental pode não representar integralmente a complexidade arquitetural e operacional encontrada em sistemas reais de produção.

Na dimensão técnica, os mecanismos de segurança implementados no ecossistema laboratorial focaram-se na mitigação dos vetores de *Cross-Site Scripting (XSS)* modelados, não prevenindo de forma nativa outras categorias de falhas em aplicações web. A aplicação da diretiva de segurança *HttpOnly*, por exemplo, impede a leitura de dados sensíveis via *scripts* do lado do cliente e o acesso direto ao *cookie* de navegação. Contudo, esse controle isolado não protege a aplicação contra ataques de *Cross-Site Request Forgery (CSRF)*, nos quais um agente malicioso induz o navegador do usuário autenticado a submeter requisições indesejadas para as rotas da API, aproveitando-se do envio automatizado das credenciais anexadas pelo próprio navegador.

Adicionalmente, técnicas baseadas em manipulação de estado, como o *Cookie Sandwich*, evidenciam limitações de escopo na persistência de sessões. Caso a infraestrutura aceite parâmetros de *cookies* injetados antes do processo de autenticação legítimo ou apresente brechas de fixação de sessão, um atacante pode cercar o *cookie* válido de autenticação com dados maliciosos gerados para subdomínios ou caminhos de URL específicos, forçando o servidor a processar

credenciais controladas.

A dependência do protocolo de transporte também se configura como uma limitação; se a *flag Secure* não estiver configurada no cabeçalho do *cookie*, as informações trafegam em texto puro, permitindo a interceptação de dados por ataques de *Man-In-The-Middle (MITM)*. Por fim, mesmo a Política de Segurança de Conteúdo (CSP) adotada pode sofrer tentativas de evasão caso ocorram injeções de *payloads* que reaproveitem elementos ou atributos HTML nativos já autorizados pela origem do próprio sistema (*self*), executando códigos a partir de manipuladores de eventos embutidos.

Portanto, o ecossistema laboratorial desenvolvido consolida-se como uma ferramenta didática replicável e eficiente, com potencial de integração a disciplinas de segurança de software em cursos de graduação em Computação e áreas afins. A evolução da ferramenta poderá abordar outros pontos não endereçados no escopo deste trabalho.

6.2 Trabalhos Futuros

Como proposta para a continuidade e ampliação desta pesquisa, sugerem-se os seguintes desdobramentos técnicos e pedagógicos:

- **Implementação e validação do modo seguro:** realizar uma nova etapa de aplicação laboratorial com os estudantes após a ativação completa dos mecanismos defensivos incorporados à arquitetura, como funções de sanitização, tratamento seguro de propriedades como *dangerouslySetInnerHTML* e aplicação de políticas de *Content Security Policy (CSP)*. A proposta visa comparar estatisticamente a evolução do aprendizado entre os cenários de exploração das vulnerabilidades e os processos de mitigação e defesa;
- **Expansão do catálogo de vulnerabilidades:** incorporar ao ecossistema de *e-commerce* outros vetores de ataque críticos presentes no OWASP Top 10, incluindo vulnerabilidades como *SQL Injection*, falhas de autenticação e quebra de controle de acesso, ampliando o escopo pedagógico e técnico da plataforma; por estar relacionado ao uso do *cookie* de sessão, ataque do tipo CSRF (*Cross-Site Request Forgery*) podem perfeitamente serem abordado como trabalhos futuros, também.
- **Desenvolvimento de um módulo de avaliação automatizada:** implementar um painel administrativo voltado ao acompanhamento do desempenho dos estudantes em tempo real, permitindo ao docente monitorar atividades, propor desafios progressivos e fornecer orientações baseadas em metodologias de Capture *The Flag (CTF)*, fortalecendo a autonomia, a

interatividade e a gamificação do processo de aprendizagem.

6.3 DECLARAÇÃO DE USO DE INTELIGÊNCIA ARTIFICIAL

Em conformidade com as diretrizes de integridade científica e boas práticas acadêmicas recomendadas pela CAPES, os autores declaram que utilizaram ferramentas de inteligência artificial, incluindo *Google Gemini 3.5 Flash*, exclusivamente para auxiliar na melhoria da linguagem e legibilidade deste manuscrito. O uso dessas ferramentas foi realizado sob supervisão humana, e os autores revisaram e editaram cuidadosamente o conteúdo, assumindo total responsabilidade pelo texto final publicado.

REFERÊNCIAS

BERTOGLIO, D. D. *et al.* Weasels e a construção de conhecimento em segurança ofensiva. In: **ANAIS DO II SIMPÓSIO BRASILEIRO DE EDUCAÇÃO EM COMPUTAÇÃO (EDUCOMP 2022)**. Porto Alegre: Sociedade Brasileira de Computação (SBC), 2022. p. 109–117. ISSN 3086-0733. Disponível em: https://doi.org/10.5753/educomp.2022.19204.

De Ryck, P. **Preventing XSS in React (Part 2)**: dangerouslysetinnerhtml. 2020. Disponível em: https://pragmaticwebsecurity.com/articles/spasecurity/react-xss-part2.html. Acesso em 30 jun. 2026.

HANNOUSSE, A.; YAHIOUCHE, S.; NAIT-HAMOUD, M. C. **Twenty-two years since revealing cross-site scripting attacks**: a systematic mapping and a comprehensive survey. 2022. Disponível em: https://arxiv.org/pdf/2205.08425. Acesso em 30 jun. 2026.

KLEIN, A. **DOM Based Cross Site Scripting or XSS of the Third Kind**. 2005. http://www.webappsec.org/projects/articles/071105.shtml. Online; acesso em 22 maio 2026.

LI, T. *et al.* **A Survey on Web Application Testing**: A decade of evolution. 2024. Disponível em: https://arxiv.org/pdf/2412.10476. Acesso em 30 jun. 2026.

META OPEN SOURCE. **React**: dangerouslysetinnerhtml. 2026. Disponível em: https://react.dev/reference/react-dom/components/common#dangerously-setting-the-inner-html. . Acesso em 03 mar. 2026.

NETO, J. C. S. *et al.* A educação em desenvolvimento de software seguro é necessária na indústria de software? respostas de profissionais de um hub de tecnologia no brasil. In: **ANAIS DO XXXVIII SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES 2024)**. Porto Alegre: Sociedade Brasileira de Computação (SBC), 2024. p. 357–366. Disponível em: https://sol.sbc.org.br/index.php/sbes/article/view/30376.

NIELES, M.; DEMPSEY, K.; PILLITTERI, V. Y. **NIST Special Publication 800-12 Revision 1**: An introduction to information security. 2017. Disponível em: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf. Acesso em 30 jun. 2026.

OWASP Foundation. **Application Security Verification Standard (ASVS): 4.0.3**. 2020. Disponível em: https://owasp.org/www-project-application-security-verification-standard/. Acesso em: 22 maio 2026.

OWASP Foundation. **Session hijacking attack**. 2024. Disponível em: https://owasp.org/www-community/attacks/Session_hijacking_attack. Acesso em 30 jun. 2026.

OWASP Foundation. **OWASP Top 10 for LLM Applications 2025**. 2025. Disponível em: https://genai.owasp.org/llm-top-10/. Acesso em: 12 mar. 2026.

Owasp, O. W. A. S. P. **Top 10:2021 – The Next Generation of Application Security**. 2021. Disponível em: https://owasp.org/Top10/2021/. Acesso em: 25 fev. 2026.

Owasp, O. W. A. S. P. **Top 10:2025 – Anticipated Risks and Trends**. 2025. Disponível em: https://owasp.org/Top10/2025/. Acesso em: 25 fev. 2026.

QUEIROZ, P. S. de; CAVALCANTI, G. J. de C. **O impacto do suporte ferramental orientado metodológico e segurança da informação no ensino prático de cursos de desenvolvimento de software**. 2024. Disponível em: https://repositorio.ifpe.edu.br/xmlui/handle/123456789/1336. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Software) – Instituto Federal de Pernambuco (IFPE), Belo Jardim. Acesso em 31 maio 2026.

ROSA, R. *et al.* Análise empírica e comparativa de ferramentas de varredura de vulnerabilidades em aplicações web usando owasp bwa e juice shop. In: **ANAIS DA XXI ESCOLA REGIONAL DE REDES DE COMPUTAÇÕES (ERRC 2024)**. Porto Alegre: Sociedade Brasileira de Computação (SBC), 2024. p. 183–188. Disponível em: https://doi.org/10.5753/errc.2024.4689.

SANGARSU, R. R. **Advancing Web Development with Single Page Applications (SPAs)**. 2019. Disponível em: https://jsaer.com/download/vol-6-iss-3-2019/JSAER2019-6-3-284-288.pdf. Acesso em 30 jun. 2026.

W3C, W. W. W. C. **Content Security Policy Level 3**. 2026. Disponível em: https://www.w3.org/TR/2026/WD-CSP3-20260505/. Acesso em: 22 maio 2026.

XIE, J.; LIPFORD, H. R.; CHU, B. **Why do programmers make security errors?** 2011. Disponível em: https://ieeexplore.ieee.org/document/6070393. Acesso em 30 jun. 2026.

APÊNDICE A – QUESTIONÁRIO DE AVALIAÇÃO DIDÁTICA

Este apêndice apresenta o instrumento de coleta de dados de abordagem quantitativa utilizado para a validação empírica do ecossistema laboratorial, aplicado de forma anônima e voluntária aos 38 estudantes participantes da dinâmica pedagógica.

1. **Antes desta aula, qual era o seu nível de conhecimento sobre vulnerabilidades XSS?**
   [ ] Nenhum conhecimento.
   [ ] Apenas teórico (já ouvi falar).
   [ ] Intermediário (conhecia os tipos, mas não como explorar/mitigar).
   [ ] Avançado (já havia implementado defesas na prática).
2. **Antes desta demonstração, você já tinha visto um ataque de XSS ser executado "ao vivo" em uma aplicação moderna?**
   [ ] Sim.
   [ ] Não, apenas conhecia o conceito teórico.
   [ ] Nunca tinha ouvido falar sobre como o ataque ocorre.
3. **Qual o maior risco que você percebeu em um ataque de XSS após ver a dinâmica?**
   [ ] Roubo de cookies e sequestro de sessão do usuário.
   [ ] Alteração visual da página (*deface*) para exibir mensagens falsas.
   [ ] Redirecionamento do usuário para sites maliciosos.
   [ ] Outro.
4. **O uso da ferramenta didática ajudou você a entender que *frameworks* modernos (como o *React*) não são 100% seguros sozinhos?**
   [ ] Sim, ficou claro que o erro humano na implementação pode criar falhas graves.
   [ ] Sim, mas achei que o *framework* protegia mais do que o demonstrado.
   [ ] Não, ainda acho que o *framework* deveria barrar tudo automaticamente.
5. **Como você avalia a ferramenta para demonstrar os perigos reais de uma codificação insegura?**
   [ ] Excelente: ver o ataque funcionando ajuda a entender a gravidade.
   [ ] Boa: ajuda a visualizar, mas gostaria de ver mais tipos de ataques.
   [ ] Regular: a teoria explicada em slides seria suficiente.
   [ ] Outro.
6. **Sobre os tipos de XSS apresentados, qual deles você considera que a aplicação didática ajudou a entender melhor? (Selecione todos que se aplicam)**

[ ] XSS Refletido: O *script* malicioso é enviado via URL/Input e retorna na resposta imediata.

[ ] XSS Armazenado: O *script* malicioso é salvo no banco de dados e executado para todos que acessarem a página.

[ ] XSS Baseado em DOM: O *script* é executado via manipulação de elementos no navegador do cliente.

7. **Como você avalia a usabilidade da ferramenta desenvolvida para fins de ensino?**

   [ ] Muito Fácil.

   [ ] Fácil.

   [ ] Difícil.

   [ ] Muito Difícil.

8. **O uso de um ambiente prático (o sistema de *e-commerce* vulnerável) facilitou a fixação do conteúdo em comparação a uma aula tradicional?**

   [ ] Sim, ver o ataque acontecendo "ao vivo" ajuda muito mais.

   [ ] Sim, ajudou a complementar a teoria.

   [ ] Não fez muita diferença.

   [ ] Outro.

9. **Com base no que foi visto, você teria interesse em aprender como aplicar as técnicas de correção (mitigação) em um próximo momento?**

   [ ] Sim, com certeza.

   [ ] Talvez, se for relevante para o meu trabalho/estágio.

   [ ] Não, prefiro focar apenas em desenvolvimento de funcionalidades.

10. **Espaço livre: Deixe uma sugestão sobre a ferramenta.**

    _____

    _____