QUALIDADE DE SOFTWARE: ESTRATÉGIAS PARA AMPLIAR A SEGURANÇA E EFICIÊNCIA EM SISTEMAS DE GESTÃO DE FROTAS

Gabriel Milanez Zarbato[1], Ana Cláudia Garcia Barbosa[2]

Resumo: O presente trabalho tem como objetivo avaliar a qualidade e a segurança de um sistema de gestão de frotas por meio da aplicação das normas ISO/IEC 25010:2023 e ISO/IEC/IEEE 21839:2019, amplamente reconhecidas na Engenharia de Software. O estudo aborda a importância da padronização de processos e da automação de testes na garantia da confiabilidade e eficiência de sistemas que manipulam dados sensíveis. A metodologia adotada inclui a realização de testes manuais e automatizados, com base em roteiros Behavior-Driven Development e ferramentas como Postman e Insomnia, além de ensaios de vulnerabilidade envolvendo ataques de força bruta, injeção de SQL e Cross-Site Scripting. Os resultados evidenciaram que, embora o sistema apresente bom desempenho funcional, foram identificadas falhas críticas relacionadas à autenticação, autorização e exposição de dados. A análise reforça a necessidade da integração entre práticas de qualidade e segurança desde o início do desenvolvimento, conforme o princípio de security by design. Conclui-se que a adoção de normas internacionais e a automação de testes constituem elementos fundamentais para assegurar a robustez, confiabilidade e proteção de sistemas corporativos sensíveis.

Palavras-chave: qualidade de software, iso/iec 25010, iso/iec 12207, segurança da informação, gestão de frotas e automação de testes.

[1]Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), gabrielzarbatto1214@gmail.com

[2]Orientador. Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), agb@unesc.net

ABSTRACT: This study aims to evaluate the quality and security of a fleet management system through the application of the ISO/IEC 25010:2011 and ISO/IEC 12207 standards, which are widely recognized in Software Engineering. The research highlights the importance of process standardization and test automation to ensure the reliability and efficiency of systems that handle sensitive data. The adopted methodology includes the execution of manual and automated tests based on BDD (Behavior-Driven Development) scripts and tools such as Postman and Insomnia, in addition to vulnerability assessments involving brute-force attacks, SQL injection, and XSS. The results showed that although the system performs well functionally, critical flaws were identified in authentication, authorization, and data exposure. The analysis reinforces the need to integrate quality and security practices from the early stages of development, according to the security by design principle. It is concluded that the adoption of international standards and test automation are essential elements to ensure the robustness, reliability, and protection of sensitive corporate systems.

Keywords: software quality, iso/iec 25010. iso/iec 12207, information security, fleet management and test automation.

1 INTRODUÇÃO

No contexto da Engenharia de Software, a busca por produtos digitais eficientes e confiáveis é um dos principais objetivos das organizações de tecnologia. Garantir que o software atenda às expectativas dos usuários e aos requisitos técnicos estabelecidos exige a adoção de padrões e critérios bem definidos. Nesse sentido, compreender o conceito de qualidade de software torna-se essencial para orientar o desenvolvimento e a avaliação de sistemas computacionais.

Inicialmente, é fundamental compreender o conceito de Qualidade de Software. De acordo com a norma ISO/IEC 25010:2011, esse conceito está relacionado a modelos utilizados na avaliação da qualidade de sistemas de software, considerando características como eficiência, usabilidade, confiabilidade, manutenibilidade e portabilidade.

Para aprimorar a qualidade dos produtos digitais, foi instituída, em 1990, a norma técnica ISO/IEC 12207, amplamente utilizada até os dias atuais em diversas empresas do setor de tecnologia. Essa norma representou um marco significativo na Engenharia de Software, ao estabelecer regras e diretrizes para o ciclo de vida do software, abrangendo desde

a concepção até a manutenção do produto. Antes de sua implementação, o processo de desenvolvimento de software não possuía padronização, apresentando métodos pouco estruturados e práticas não consolidadas, o que resultava em atrasos frequentes, custos elevados e comprometimento da qualidade final do produto.

Os benefícios decorrentes da adoção desta norma tornaram-se rapidamente evidentes para as organizações que a implementaram. A implementação da ISO/IEC 12207 resultou na criação de processos de desenvolvimento de software claramente definidos e bem estruturados. Essa definição detalhada dos processos aumentou consideravelmente a previsibilidade dos projetos, permitindo melhor estimativa de prazos, custos e recursos necessários. Adicionalmente, a norma contribuiu para o aumento da confiabilidade dos produtos de software, garantindo um desempenho mais estável e consistente. Outro benefício foi o aumento da conformidade com os padrões de qualidade estabelecidos pela indústria, assegurando que os produtos atendessem a requisitos rigorosos.

Um dos aspectos mais relevantes da ISO/IEC 12207 é a incorporação das práticas de verificação e validação ao longo de todas as etapas do ciclo de vida do desenvolvimento de software. Essas práticas asseguram que o software além de atender aos requisitos funcionais, também ofereça níveis adequados de segurança, protegendo dados e sistemas contra possíveis ameaças e vulnerabilidades. Ao longo do tempo, a ISO/IEC 12207 ganhou ampla aceitação na comunidade de desenvolvimento de software, tornando-se uma base sólida e reconhecida para o estabelecimento de normas mais recentes e avançadas. Atualmente, a ISO/IEC 12207 é reconhecida como um referencial fundamental que orienta o desenvolvimento e contribui de forma significativa para a evolução contínua da engenharia de software. Em particular, normas como a ISO/IEC 25010, que aborda a qualidade do produto de software, são fortemente influenciadas e construídas sobre os princípios estabelecidos pela ISO/IEC 12207. A relevância e o impacto duradouro desta norma demonstram a sua importância fundamental para a garantia da qualidade e a evolução das etapas de desenvolvimento de software.

Consequentemente, a aplicação de testes de software aliada ao uso de ferramentas apropriadas e ao trabalho de uma equipe capacitada desempenha um papel fundamental em diferentes segmentos do mercado, como por exemplo nas aplicações de gestão de frotas, no qual será o foco deste projeto (Moroz, 2020). Nesses contextos, a qualidade do software e

a segurança das informações se destacam como elementos cruciais para assegurar a otimização e a confiabilidade operacional dos sistemas. Assim, a adoção de automação de testes e de métricas de qualidade torna-se indispensável para manter o funcionamento adequado das funcionalidades críticas e proteger dados sensíveis, isto é, dados pessoais, como nome, e-mail, informações de cobrança como cartões de crédico e endereços.

Com o inevitável avanço da tecnologia, a crescente digitalização de processos, como o envio de documentos de maneira *online*, trouxe consigo desafios consideráveis, principalmente no que se diz respeito à segurança de dados em sistemas de frota. Tais plataformas lidam com informações altamente sensíveis, como localização em tempo real, documentos e dados de autenticação de usuários, dentre eles fotos, nome completo, RG, CPF e CNH. Estas informações são vulneráveis e alvos de pessoas mal-intencionadas (mais comumente chamadas de crackers), que se aproveitam de brechas de segurança de um sistema para realizarem ataques cibernéticos e, se não forem tratados da maneira correta, como afirmam Wang, H., Zhang, Y., e Ma, C. (2020), o vazamento destas informações pode acarretar em enormes prejuízos tanto para os usuários quanto para a empresa responsável pelo aplicativo. Tendo isso em mente, evidencia-se a necessidade de ferramentas e práticas que integrem qualidade e segurança desde as etapas iniciais do desenvolvimento de software, pois, como afirmam (Grimm; Sax, 2022), soluções baseadas em gráficos de conhecimento e modelos orientados a objetos podem oferecer medidas de segurança mais eficazes para lidar com esses desafios durante o desenvolvimento do projeto e após o lançamento, garantindo assim uma longevidade maior no mercado.

Além disso, a ausência de processos claros e bem formulados para avaliação da qualidade de software compromete o resultado final, ou seja, a eficiência dos testes e a detecção de vulnerabilidades. Conforme citado anteriormente, a recomendação da norma ISO/IEC 25010:2011 deixa explícito que, aspectos como confiabilidade, eficiência e usabilidade devem ser avaliados para garantir que o produto final atenda aos padrões desejados. Nesse sentido, ferramentas como o Postman e Insomnia têm se destacado na atuação de testes de API, no qual utiliza-se de *endpoints* do sistema, no qual nos permite realizar ações no sistema sem utilizar o *front-end* da aplicação, possibilitando assim a validação de regras de negócio, como permissões de *CRUD*, de forma eficiente e em larga escala (Tusing; Brooks, 2023).

Portanto, o problema central deste estudo reside em propor estratégias que combinem automação de testes e análise de métricas para melhorar a qualidade do software em sistemas de gestão de frotas e, para mapear sua utilidade, as ferramentas e técnicas descritas aqui estão sendo aplicadas em um sistema desenvolvido no ano de 2024 na disciplina de Engenharia de Software II, no qual possui uma parceria com uma empresa *startup* de gestão de frotas com o nome de *Drivers Company*. No protótipo, estão sendo utilizadas tecnologias tais como Typescript, Javascript, Tailwind CSS e utilizando o banco de dados Postgres. Essa abordagem visa proteger dados sensíveis contra possíveis ataques, mas também assegurar que o software entregue esteja alinhado aos requisitos de confiabilidade, eficiência e segurança, atendendo às necessidades operacionais e às expectativas do mercado.

Com base nesse cenário, torna-se essencial para compreender as melhores práticas e normas aplicadas ao desenvolvimento de sistemas de gestão de frotas, incluindo também implementar soluções práticas que permitam avaliar a eficácia dessas abordagens frente aos riscos reais de qualidade e segurança. Assim, este trabalho propõe a realização de um conjunto estruturado de testes, incluindo técnicas baseadas na ISO/IEC 25010:2011 e na ISO/IEC 12207, tal como testes de força bruta, quebras de senha e injeção de código SQL malicioso, além do escaneamento da aplicação web para identificação de vulnerabilidades. A análise desses aspectos será consolidada na seção de resultados e discussão, visando documentar as falhas encontradas, com finalidade de propor alternativas para mitigação de riscos e fortalecimento da segurança do sistema.

A qualidade de software e a segurança de dados são temas centrais no desenvolvimento de sistemas, especialmente em ambientes que lidam com informações sensíveis, como os sistemas de gestão de frotas. A seguir, apresentam-se os principais fundamentos que sustentam este trabalho, organizados em tópicos, estes são: Normas ISO Relacionadas à Qualidade e Segurança, Qualidade de Software, Automação de Testes de Software e Segurança da Informação em Sistemas de Gestão de Frotas.

1.1 NORMAS ISO RELACIONADAS À QUALIDADE E SEGURANÇA

Como sugere a ISO 27001 e ISO 9001, o aumento da digitalização e a complexidade das aplicações exigem um amplo refinamento na abordagem de práticas rigorosas para garantir a eficiência, confiabilidade e segurança das informações. Em conjunto, estas duas normas têm um

enorme destaque dentro do escopo mencionado. De acordo com (Fiore, 2020), a integração dos sistemas de gestão da segurança da informação, conforme a ISO 27001, com os sistemas de gestão da qualidade, conforme a ISO 9001, é essencial para otimizar recursos e melhorar a eficácia organizacional, sendo necessário identificar e priorizar os fatores que influenciam essa integração, a fim de guiar a alocação eficiente de recursos (Fiore, 2020).

Além disso, Kaluvakuri e Khambam (2024) destacam a importância da integração entre sistemas de gestão de identidade e acesso (IAM) e modelos de aprendizado de máquina para reforçar a integridade dos dados em aplicações de gestão de frotas baseadas na nuvem, evidenciando como o cumprimento de normas e a inovação tecnológica são aliados fundamentais na proteção da informação (Kaluvakuri; Khambam, 2024).

1.2 QUALIDADE DE SOFTWARE

A qualidade de software é um atributo essencial que define a capacidade de um sistema em atender aos requisitos funcionais e não funcionais estabelecidos. De acordo com a ISO/IEC 25010:2011, qualidade de software abrange dimensões como funcionalidade, usabilidade, eficiência, segurança e manutenibilidade. A implementação de padrões internacionais melhora a experiência do usuário e reduz falhas operacionais, enquanto a não conformidade com esses requisitos compromete diretamente a segurança e a confiabilidade do sistema.

Falco, Núñez e Tanzi (2019) afirmam que, embora o desenvolvimento de plataformas modernas de monitoramento de frotas com integração de *IoT* tenha avançado, a segurança muitas vezes depende exclusivamente de recursos nativos dos *frameworks*, o que evidencia a necessidade de adoção de práticas adicionais de qualidade e segurança (Falco; Núñez; Tanzi, 2019).

1.3 AUTOMAÇÃO DE TESTES DE SOFTWARE

A automação de testes desempenha um papel fundamental no desenvolvimento de sistemas seguros e confiáveis. De acordo com Hildélio Júnior (2025), a automação impacta diretamente na qualidade, nos custos e no tempo de entrega dos produtos, permitindo detectar defeitos em fases iniciais do ciclo de desenvolvimento e reduzindo significativamente os custos relacionados à correção de *bugs* (Júnior, 2025).

Haj Hamad (2018) reforça essa perspectiva ao propor um labora-

tório automatizado para monitoramento e testes em sistemas de gestão de frotas baseados em *IoT*, demonstrando que a automação não apenas reduz falhas, mas também aumenta a eficiência e a segurança nesses ambientes críticos (Hamad, 2018). Além disso, Takanen et al. (2018) ressaltam que técnicas como o *fuzzing* são ferramentas essenciais para identificar falhas críticas de segurança em sistemas complexos, funcionando como uma das metodologias mais eficazes dentro do escopo de testes automatizados (Takanen et al., 2018).

1.4 SEGURANÇA DA INFORMAÇÃO EM SISTEMAS DE GESTÃO DE FROTAS

Os sistemas de gestão de frotas armazenam dados como localização em tempo real, documentos de identificação e informações financeiras. A vulnerabilidade dessas informações a ataques cibernéticos torna essencial a adoção de estratégias de proteção, como criptografia e autenticação multifator. Estudos recentes destacam que falhas na segurança podem resultar em prejuízos financeiros e danos à reputação da empresa. A segurança deve ser incorporada desde a fase de desenvolvimento do software, seguindo o conceito de "*Security by Design*", que inclui testes de segurança, revisão de código e monitoramento contínuo para detecção de ameaças.

Pesquisadores enfatizam que sistemas que adotam essas práticas apresentam uma redução significativa nos incidentes de segurança. Um artigo publicado pela (Security Leaders; Kaspersky, 2024), em parceria com a Kaspersky, aponta que:

> Além da maioria das empresas brasileiras terem sofrido ao menos um incidente de segurança nos últimos anos, a pesquisa mostra também que 58 porcento desses ataques foram classificados como graves, o que pode resultar em perdas financeiras, danos à reputação e interrupção dos negócios. A crescente sofisticação dos ataques exige que as organizações invistam em uma estratégia de cibersegurança robusta e em programas de conscientização para seus funcionários.

Complementarmente, Hamadaqa, Mars e Adi (2019) propõem a adoção de arquiteturas que utilizam entidades físicas resistentes à clonagem como uma camada adicional de segurança em sistemas de gestão de frotas, evidenciando que estratégias físicas e lógicas devem ser integradas para uma proteção mais abrangente (Hamadaqa; Mars; Adi, 2019).

No geral, o objetivo foi em validar e garantir a qualidade de um sistema de gestão de frotas com base na norma ISO/IEC 25010:2011, com ênfase na verificação e análise da segurança das informações.

Os objetivos específicos deste trabalho concentram-se na aplicação de boas práticas e normas de qualidade no desenvolvimento e validação do sistema de frota. Primeiramente, busca-se aplicar a norma ISO/IEC 25010:2011 na avaliação da qualidade do sistema, garantindo que os critérios estabelecidos sejam atendidos. Além disso, serão utilizadas técnicas de teste de software voltadas para assegurar a segurança e a integridade dos dados. Outro ponto fundamental é a adoção de metodologias de qualidade voltadas para APIs, com foco em segurança e desempenho. Também será realizada a automação de testes utilizando o Postman, de modo a validar permissões de CRUD e efetuar testes de performance. Por fim, pretende-se avaliar a integração contínua e a segurança dos dados, garantindo a execução de testes automáticos que reforcem a qualidade do sistema.

2 TRABALHOS CORRELATOS

A situação demonstrada tem um grande impacto tanto econômico quanto social. Tendo isso em vista, pode-se concluir que existam soluções similares ao problema elaborado. Diversos estudos abordam aspectos relacionados à segurança e qualidade de software em sistemas de gestão de frotas, bem como o uso de automação de testes. Haj Hamad et al. (Hamad, 2018) propuseram uma plataforma de testes automatizados na nuvem para sistemas inteligentes de gestão de frotas baseados em IoT. Utilizaram ferramentas como Selenium e UFT para automação de testes de APIs, integrando segurança como parte essencial do desenvolvimento. Embora relevante, o foco está em soluções baseadas em ambientes na nuvem, enquanto este trabalho aplica-se a um contexto prático de desenvolvimento com tecnologias modernas como TypeScript e Postgres, com enfoque em testes de segurança manuais e roteiros de implementação para testes automatizados. Pargaonkar (Pargaonkar, 2023) realizou uma revisão abrangente sobre metodologias de testes de segurança e tendências emergentes na engenharia da qualidade de software. O autor destaca desde práticas tradicionais até técnicas como *fuzzing* e análise comportamental, salientando a importância da automação para detecção precoce de vulnerabilidades. Diferentemente, este trabalho vai além da revisão teórica, promovendo uma aplicação prática e orientada à segurança de APIs em um

sistema de gestão de frotas real. Hamadaqa et al. (Hamadaqa; Mars; Adi, 2019) propuseram uma arquitetura de segurança física para sistemas de gestão de frotas, utilizando entidades resistentes à clonagem para proteção contra acessos não autorizados. Embora pertinente, a proposta centra-se na segurança física e de hardware, enquanto este trabalho foca na segurança lógica, com destaque para a proteção de dados sensíveis via práticas de segurança cibernética e automação de testes. Dessa forma, observa-se que, embora existam propostas relevantes voltadas à segurança e qualidade de software em sistemas de gestão de frotas, bem como à automação de testes, nenhuma delas contempla de maneira integrada a aplicação prática de roteiros de testes manuais e automatizados com foco específico na segurança de APIs em um contexto real. Assim, o presente trabalho diferencia-se por unir essas abordagens, contribuindo de forma positiva para o fortalecimento da segurança e da qualidade em sistemas críticos de gestão de frotas.

3 MATERIAIS E MÉTODOS

A metodologia do trabalho esteve estruturada de modo a garantir uma abordagem prática e sistemática para a avaliação da qualidade e da segurança do sistema de gestão de frotas, fundamentada nas diretrizes da norma ISO/IEC 25010:2011 e na aplicação de técnicas de teste de software alinhadas ao ciclo de vida de desenvolvimento definido pela ISO/IEC 12207.

O processo de testes foi desenvolvido em diferentes etapas. O primeiro passo consistiu na preparação do ambiente, que envolveu a configuração da aplicação web desenvolvida na disciplina de Engenharia de Software II, destinada à Drivers Company, utilizando as tecnologias TypeScript, JavaScript, Tailwind CSS e banco de dados SQL. Para suportar a execução dos testes, foram disponibilizados endpoints de API com foco nas operações de CRUD (Create, Read, Update e Delete). A aplicação foi hospedada em uma Máquina Virtual utilizando o Oracle VirtualBox, enquanto ferramentas como o Postman, voltado para testes de API, e o Insomnia, utilizado como suporte adicional para requisições e análises de resposta, foram devidamente instaladas e configuradas.

Na segunda etapa, foram implementados testes automatizados a partir da criação de roteiros funcionais que seguiram a lógica de negócios do sistema. Esses testes foram conduzidos utilizando o Postman, abrangendo os principais fluxos da aplicação, complementados por testes manuais realizados com interação direta. As validações contemplaram permis-

sões de acesso (CRUD), autenticação e autorização, performance e tempo de resposta dos endpoints, além da integridade dos dados manipulados pelas APIs. Também foi aplicada a abordagem BDD (Behavior-Driven Development), que permitiu estruturar os roteiros de teste de forma a possibilitar sua futura conversão em suítes automatizadas, ampliando a cobertura de qualidade.

A terceira fase correspondeu à execução de testes manuais de segurança, cujo objetivo foi identificar vulnerabilidades técnicas com alto potencial de exploração. Entre as técnicas aplicadas, destacaram-se testes de força bruta para validação de credenciais, tentativas de quebra de senha utilizando ferramentas especializadas, injeção de código SQL malicioso, ataques de negação de serviço (DoS/DDoS) em ambiente controlado, escaneamento de vulnerabilidades com auxílio de ferramentas de análise, verificação de versões desatualizadas de bibliotecas, frameworks e banco de dados, além da busca por arquivos expostos e configurações de segurança inadequadas. Cada vulnerabilidade identificada foi classificada conforme seu grau de severidade e impacto potencial.

Na sequência, os dados obtidos durante a execução dos testes foram organizados em relatórios detalhados. Cada ocorrência foi descrita com informações como nome e descrição da vulnerabilidade ou falha de qualidade, técnica utilizada para identificação do problema, riscos associados, recomendações de correção e evidências documentadas por capturas de tela e logs de execução. A análise dos resultados foi realizada em conformidade com os critérios da norma ISO/IEC 25010:2011, de forma a relacionar as falhas identificadas aos atributos de qualidade, como segurança, confiabilidade e eficiência.

Por fim, foi elaborado um relatório técnico consolidado, contendo os resultados da análise, a classificação dos riscos, as recomendações de segurança e um plano de ação direcionado à mitigação das falhas funcionais e vulnerabilidades encontradas na aplicação, garantindo, assim, maior robustez e qualidade ao sistema avaliado.

A execução do trabalho exigiu a disponibilização de recursos materiais (hardware e software) e humanos devidamente estruturados para atender às necessidades do projeto. No que se refere ao hardware, foram utilizados dois computadores com processador Quad-Core, 8 GB de memória RAM e disco rígido de 100 GB cada, ou, alternativamente, um computador com processador Quad-Core, 16 GB de memória RAM e disco rígido de 200 GB.

Quanto ao software, os sistemas operacionais empregados foram o Linux Debian (Kali Linux) e o Windows 10, ambos em arquitetura x64. Além disso, a aplicação de Gestão de Frotas Web fez parte fundamental do ambiente, acompanhada por uma base de dados estruturada em SQL. Para o suporte à criação e execução de testes, foram utilizados o Gherkin em conjunto com o Cucumber, o Postman para automação e validação de APIs, e o Oracle VirtualBox para a configuração de máquinas virtuais necessárias ao ambiente de testes.

4 RESULTADOS E DISCUSSÃO

Os testes funcionais executados no sistema de gestão de frotas evidenciaram a conformidade das principais funcionalidades do protótipo desenvolvido, abrangendo desde a navegação inicial até operações críticas de CRUD (Create, Read, Update, Delete) sobre motoristas e veículos. Os roteiros estruturados permitiram validar aspectos essenciais como autenticação, funcionalidades de pesquisa, listagem, cadastro, visualização e exclusão de registros, além de fluxos administrativos específicos, tais como aprovação de motoristas pendentes e emissão de alertas do sistema.

De forma geral, as funcionalidades atenderam às especificações estabelecidas, apresentando formulários com validações adequadas e elementos de interface responsivos. Contudo, foram identificados pontos de melhoria em cenários de usabilidade, particularmente no que se refere à necessidade de feedback mais explícito em situações de erro de autenticação e aprimoramento das mensagens de validação apresentadas ao usuário. Concluindo, as técnicas utilizadas são amplamente evidenciadas como estratégias de melhorias de projetos, visto que vemos empresas cada dia mais implementando tais procedimentos na vida útil de seus softwares

4.1 ANÁLISE DE SEGURANÇA

Em paralelo à avaliação funcional, a auditoria de segurança revelou vulnerabilidades de alto impacto que comprometem significativamente a integridade do sistema. Entre os resultados mais críticos, destacaram-se: (i) falha na implementação da autenticação devido ao uso de senha fixa no código-fonte; (ii) ausência de mecanismos de autorização em endpoints administrativos; e (iii) exposição de dados pessoais sensíveis sem qualquer camada de proteção adequada.

Esses fatores comprometem diretamente os requisitos de segurança estabelecidos pela norma ISO/IEC 25010:2011, especialmente nas

dimensões de confidencialidade, integridade e disponibilidade. A análise demonstrou também que os cookies de sessão estavam configurados de forma inadequada, sem a implementação de flags de proteção essenciais, além da ausência de mecanismos de prevenção contra ataques do tipo CSRF (Cross-Site Request Forgery), força bruta ou limitação de taxa de requisições (rate limiting).

Complementarmente, foram identificadas falhas de configuração críticas, incluindo a ausência de cabeçalhos de segurança essenciais como Content Security Policy (CSP), HTTP Strict Transport Security (HSTS) e X-Frame-Options (XFO), bem como a presença de credenciais com baixo nível de complexidade em arquivos de ambiente, fatores que amplificam significativamente os riscos de exploração por agentes maliciosos.

4.2 CORRELAÇÃO COM A LITERATURA CIENTÍFICA

A comparação dos resultados obtidos com a literatura especializada abordada neste trabalho demonstra que os problemas identificados não constituem casos isolados. Estudos recentes citados neste artigo, já destacam que aplicações voltadas à gestão de frotas frequentemente negligenciam a implementação de camadas robustas de segurança, apoiando-se exclusivamente em recursos nativos dos frameworks de desenvolvimento utilizados.

No presente caso, a inexistência de políticas adequadas de autorização e a exposição das vulnerabilidades identificadas alinham-se às falhas documentadas em pesquisas anteriores, reforçando a necessidade imperativa de adoção do princípio de *security by design* desde as fases iniciais do desenvolvimento. Esse cenário evidencia que, sem diretrizes claras e mecanismos eficazes de proteção, o sistema permanece suscetível a riscos que poderiam ser mitigados com práticas mais maduras de engenharia de software.

Adicionalmente, os resultados da auditoria confirmam a relevância da automação de testes de segurança, conforme defendido por autores que correlacionam práticas de Behavior-Driven Development (BDD) e ferramentas especializadas como Postman e Cucumber à redução significativa de riscos e à ampliação da cobertura de qualidade em sistemas de informação. Tais ferramentas tornam o processo de validação mais consistente e facilitam a detecção precoce de falhas que, em fases posteriores, poderiam demandar maior esforço de correção.

Na Figura 1 abaixo está um exemplo de uma escrita de teste

de API automatizado utilizando o framework Cypress, ilustrando como a automação auxilia na verificação contínua das funcionalidades e reforça a confiabilidade do sistema.

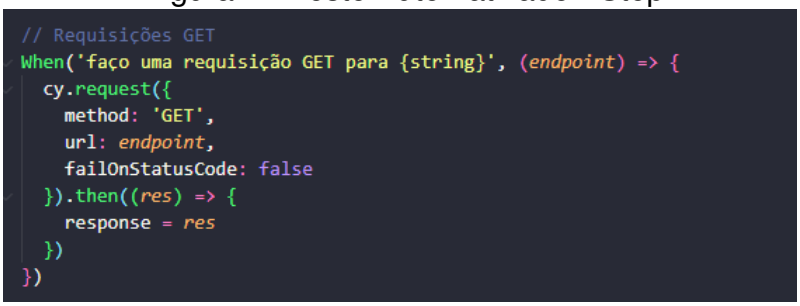
Figura 1 - Teste Automatizado - Gherkin

```
Funcionalidade: API de Motoristas
  Como um sistema cliente
  Eu quero consumir a API de motoristas
  Para integrar com outros sistemas

  Contexto:
    Dado que a API está disponível em "http://localhost:3000/api"

  Cenário: Listar todos os motoristas via API
    Quando faço uma requisição GET para "/api/driver"
    Então recebo status 200
    E recebo uma lista de motoristas
    E cada motorista contém:
      | Campo         | Tipo    |
      | id            | number  |
      | cnh           | string  |
      | approved      | boolean |
      | user.name     | string  |
      | user.cpf      | string  |
      | user.email    | string  |
```

Fonte: Do autor

Conforme a figura 1, temos o escopo no qual o teste está implementado, seguido pelo cenário, finalizando pelas respostas esperadas (podem ser campos contendo valores específicos, quanto códigos de resposta de endpoints). Para a linguagem Gherkin poder funcionar, é necessária a criação de steps em alguma linguagem de programação (exemplos nas Figuras 2 e 3 com JavaScript + Cypress):

Figura 2 - Teste Automatizado - Step



```
// Requisições GET
When('faço uma requisição GET para {string}', (endpoint) => {
  cy.request({
    method: 'GET',
    url: endpoint,
    failOnStatusCode: false
  }).then((res) => {
    response = res
  })
})
```

Fonte: Do autor

When é a definição de uma validação seguida por uma breve descrição de sua função, no qual irá receber um endpoint. Em cy.request, é onde passamos os parâmetros de testes para que a automação aprove o cenário. Qualquer divergência, é retornado um erro, conforme a Figura 3.

Figura 3 - Teste Automatizado - Utilização do comando

```
// Comando para login rápido
Cypress.Commands.add('loginAsAdmin', () => {
    // Login direto via API (sem UI)
  cy.request({
    method: 'POST',
    url: '/api/user/login/web',
    body: {
      email: 'gabriel@admin.com',
      password: 'admin'
    }
  }).then((response) => {
    //A resposta esperada deve ser 200 OK, caso contrário o teste falha
    expect(response.status).to.eq(200)
    // Caso o login utilize cookies de sessão, poderá ser definido aqui
    cy.setCookie('email', 'gabriel@admin.com')
  })
})
```

Fonte: Do autor

No geral, temos a mesma estrutura de dados e parâmetros. Foi definido também os dados de login, para que o framework possa acessar a aplicação de maneira autônoma. Adicionalmente, o mais correto seria inserir os dados de acesso em um arquivo controlado (que não é enviado para ambientes de acesso público) para não se correr o risco de vazamentos destas informações. Foi inserido desta forma para facilitar o entendimento.

4.3 ANÁLISE DAS VULNERABILIDADES CRÍTICAS ENCONTRADAS

A análise do código-fonte da aplicação revelou a presença de uma constante de senha fixada na classe responsável pela autenticação de usuários. Esta implementação inadequada faz com que o sistema ignore completamente a senha vinculada ao usuário armazenada no banco de dados, aceitando qualquer valor inserido no campo de senha durante o processo de autenticação. Conforme na Figura 4, notamos: *const valid = password === 'admin';*. Isso define uma senha fixa para a aplicação, ignorando totalmente a informação na base de dados:

Figura 4 - Vulnerabilidades - Senha fixada no código

```
import { NextResponse } from 'next/server';
import { query } from '@/lib/db';

export async function POST(request) {
  try {
    const body = await request.json();
    const { email, password } = body;

    if (!email || !password) {
      return NextResponse.json({ error: 'Credenciais inválidas' }, { status: 400 });
    }

    const result = await query('SELECT id, email, password_hash, name, last_name FROM app_user WHERE email
    if (result.rowCount === 0) {
      return NextResponse.json({ error: 'Usuário ou senha inválidos' }, { status: 401 });
    }
    const user = result.rows[0];

    const valid = password === 'admin';
    if (!valid) {
      return NextResponse.json({ error: 'Usuário ou senha inválidos' }, { status: 401 });
    }

    return NextResponse.json({ email: user.email, name: user.name, last_name: user.last_name });
  } catch (error) {
    console.error('Erro login:', error);
    return NextResponse.json({ error: 'Erro interno' }, { status: 500 });
  }
}
```

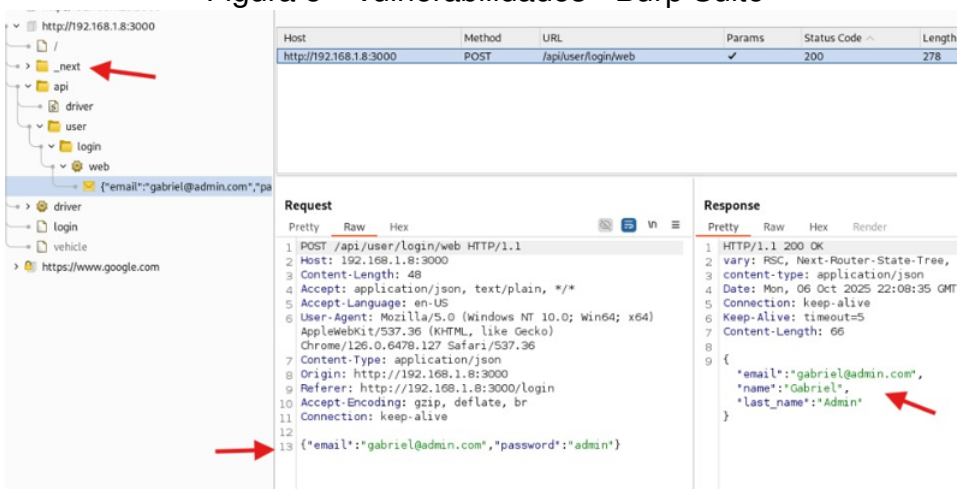
Fonte: Do autor

Esta vulnerabilidade representa um risco crítico de segurança, uma vez que um atacante que obtenha conhecimento do endereço de e-mail de qualquer usuário legítimo do sistema pode realizar acesso não autorizado, independentemente do conhecimento da senha real associada à conta.

A aplicação encontra-se hospedada em ambiente de nuvem através de endereço que não implementa criptografia HTTPS. Esta configuração inadequada resulta na ausência de certificado digital para criptografar as requisições transmitidas entre cliente e servidor, tornando o sistema vulnerável a ataques do tipo Man-in-the-Middle (MITM) e técnicas de interceptação de tráfego (sniffing).

Para demonstrar a gravidade desta vulnerabilidade, foram utilizadas ferramentas especializadas em análise de tráfego de rede: Burp Suite, para interceptação e manipulação de requisições HTTP, e Wireshark, para monitoramento detalhado do tráfego de rede. Os testes evidenciaram que todas as requisições, incluindo dados sensíveis como credenciais de autenticação, são transmitidas em texto plano, permitindo sua visualização e captura por agentes não autorizados, conforme nas Figuras 5, 6 e 7.

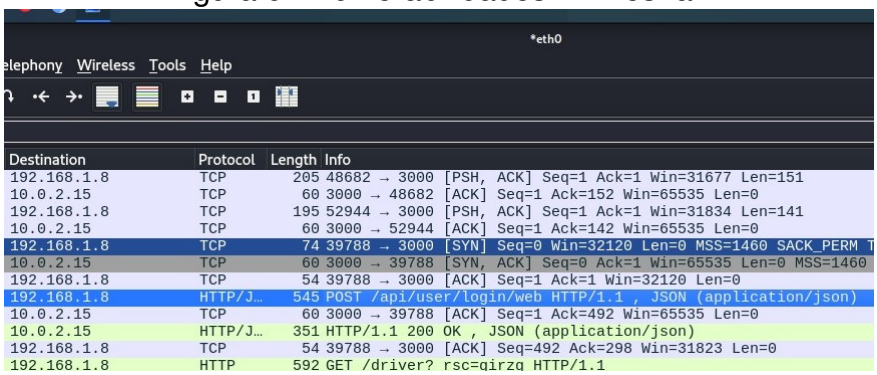
Figura 5 - Vulnerabilidades - Burp Suite



Fonte: Do autor

Na Figura 5, notamos ao lado esquerdo o armazenamento das informações ao tentarmos entrar na aplicação. Em *Request* e *Response*, vemoz as informações sensíveis visíveis.

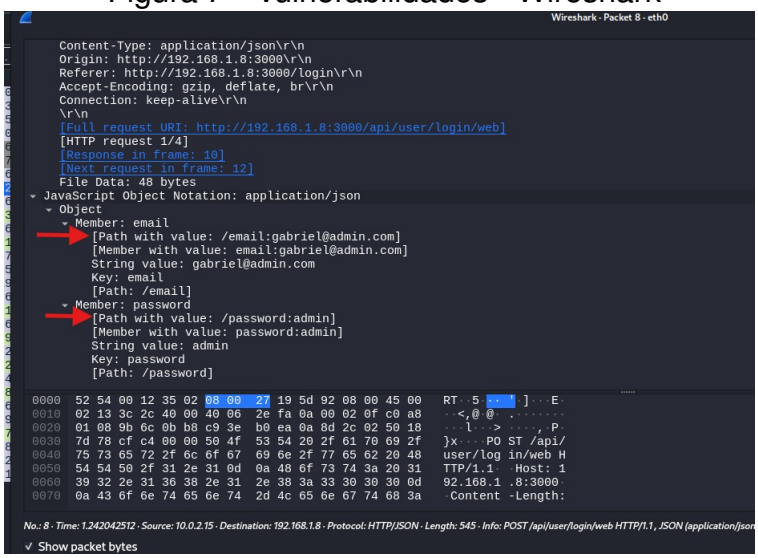
Figura 6 - Vulnerabilidades - Wireshark



Fonte: Do autor

Na Figura 6, temos o registro de tráfego com o mesmo endpoint */api/user/login/web*. Se acessarmos este registro, vemos: o email e senha, indicado pelas setas na Figura 7:

Figura 7 - Vulnerabilidades - Wireshark

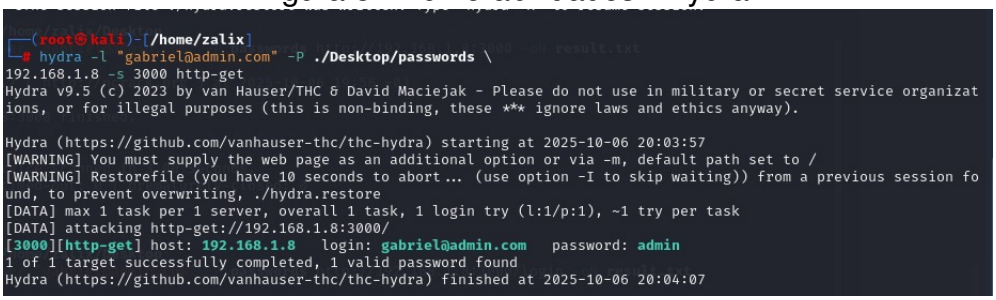


Fonte: Do autor

O sistema não implementa requisitos mínimos para criação de senhas, permitindo o cadastro de usuários com credenciais de baixa complexidade. Esta deficiência torna o sistema suscetível a ataques de quebra de senha (password cracking) utilizando ferramentas especializadas como Ncrack, John the Ripper, Medusa e Hashcat.

Para validação desta vulnerabilidade, foi empregada a ferramenta Hydra, configurada para realizar ataques de dicionário online. Os resultados, conforme a Figura 8, demonstraram que o acesso não autorizado pode ser obtido em tempo próximo a zero, evidenciando a ausência de políticas de senha forte e de mecanismos de autenticação multifator, considerados essenciais nos padrões atuais de segurança da informação.

Figura 8 - Vulnerabilidades - Hydra



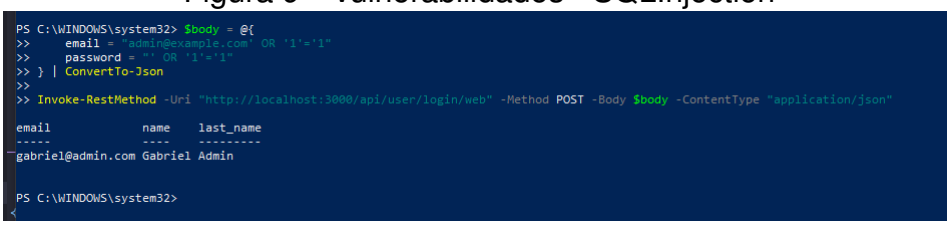
Fonte: Do autor

Observa-se que no terminal foi inserido o comando hydra (inicializa o software) -l (define um usuário único para o teste) -P (define uma lista de possíveis senhas) ./Desktop/passwords (localização da lista) 192.168.1.8 (host da aplicação) -s 3000 (porta 3000) http-get (tipo de conexão). O resultado da senha é reportado logo no final da execução, junto com o login, host, método e porta de conexão.

A análise dos formulários de entrada de dados revelou a presença de vulnerabilidades de SQL Injection, uma técnica de ataque na qual comandos SQL maliciosos são inseridos em campos de entrada da aplicação, com o objetivo de executar operações não autorizadas na base de dados.

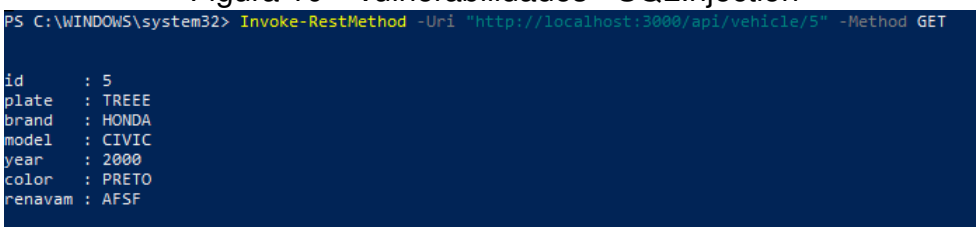
Os testes foram realizados no formulário de login de usuários, onde foi inserido o comando SQL OR '1'='1' nos campos de acesso. Este comando tem como objetivo eliminar completamente a validação da aplicação por uma senha. A sintaxe utilizada segue o padrão em que a expressão lógica OR '1'='1' força a condição a ser sempre verdadeira, permitindo a autenticação sem qualquer verificação real na base de dados. Esse tipo de teste é amplamente utilizado para demonstrar a fragilidade de sistemas que não implementam corretamente mecanismos de sanitização e validação de entrada. A execução e o resultado desse procedimento podem ser observados na Figura 9.

Figura 9 - Vulnerabilidades - SQLInjection



Fonte: Do autor

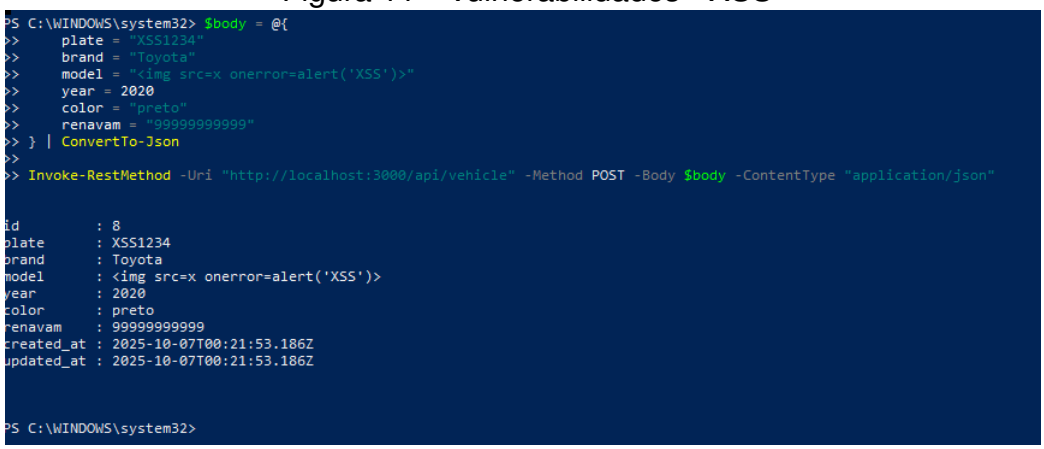
Figura 10 - Vulnerabilidades - SQLInjection



Fonte: Do autor

O sistema apresenta também vulnerabilidades de Cross-Site Scripting (XSS), uma técnica de ataque similar ao SQL Injection, porém utilizando scripts JavaScript maliciosos inseridos em campos de entrada da aplicação. Esta vulnerabilidade permite que atacantes executem código JavaScript no navegador de outros usuários, possibilitando o roubo de informações sensíveis, sequestro de sessões e redirecionamento para sites maliciosos. Na Figura 11 está uma requisição enviada pelo PowerShell para a aplicação. No campo "model"do payload da requisição, nota-se JavaScript: *<img src=x onerror=alert('XSS')>*. Isso irá fazer com que a aplicação retorne uma mensagem na aplicação quando for disparado um erro.

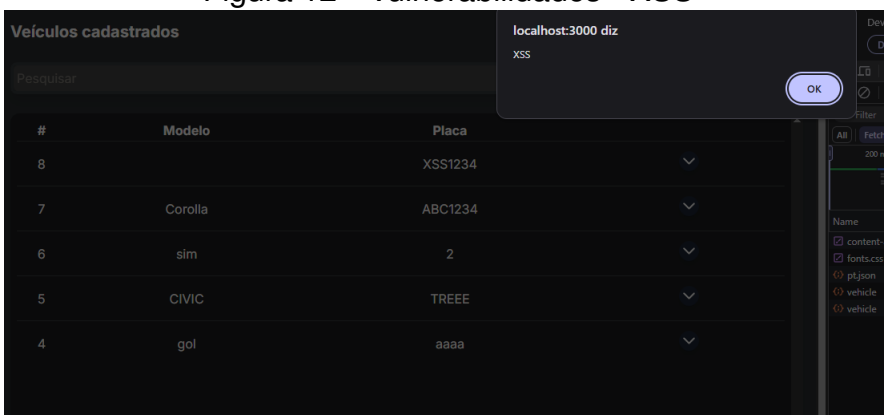
Figura 11 - Vulnerabilidades - XSS



Fonte: Do autor

Ao acessar a tela de cadastro de veículos, observamos que o script é corretamente acionado e passa a executar suas rotinas previstas, evidenciando o funcionamento esperado do mecanismo de automação. Esse comportamento pode ser verificado na Figura 12 abaixo, que ilustra de forma clara o início da execução do processo.

Figura 12 - Vulnerabilidades - XSS



Fonte: Do autor

4.4 SÍNTESE DOS RESULTADOS

Os resultados obtidos evidenciam que, embora o sistema atenda de forma satisfatória aos requisitos funcionais básicos estabelecidos, ele apresenta fragilidades em aspectos de segurança da informação. Esta dicotomia implica que a confiabilidade percebida pelo usuário final pode ser significativamente comprometida por falhas exploráveis em ambiente de produção.

A principal contribuição prática deste estudo consiste na demonstração empírica de que a simples validação funcional não é suficiente para garantir a qualidade total do software. Sem a implementação de testes direcionados especificamente à segurança da informação, o risco de vazamentos de dados e acessos não autorizados permanece em nível crítico.

Em síntese, os achados confirmam a hipótese inicial de que a aplicação de normas internacionais de qualidade, aliada à implementação de testes manuais e automatizados abrangentes, é imprescindível para assegurar eficiência operacional e proteção adequada em sistemas críticos de gestão de frotas. A correlação entre os resultados funcionais e de segurança demonstra que apenas a integração harmoniosa entre usabilidade, desempenho e mecanismos robustos de proteção pode consolidar um produto confiável e aderente às normas internacionais de qualidade de software.

5 CONCLUSÃO

A pesquisa atingiu seus objetivos ao aplicar as normas ISO/IEC 25010:2011 e ISO/IEC 12207 na avaliação da qualidade e segurança de um sistema de gestão de frotas. Os testes funcionais comprovaram estabilidade nas operações principais, enquanto a auditoria revelou falhas críticas

em autenticação, autorização e exposição de dados. As contribuições incluem a aplicação prática de testes manuais e automatizados e a identificação de vulnerabilidades reais com propostas de correção. Como limitações, destacam-se o ambiente restrito e a ausência de integração contínua.

Recomenda-se, para trabalhos futuros, ampliar a automação dos testes, integrar pipelines de segurança contínua e aplicar práticas de security by design para maior robustez, ou a implementação de um novo sistema, contendo proteção das vulnerabilidades citadas.

REFERÊNCIAS

FALCO, M.; NúñEZ, I.; TANZI, F. Improving the fleet monitoring management, through a software platform with iot. In: **IEEE International Conference**. [s.n.], 2019. Acesso em: 02 maio 2025. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8980429/>.

FIORE, A. P. **A integração de sistemas de gestão da segurança da informação com sistemas de gestão da qualidade**. 2020. Artigo para UNESP.

GRIMM, S.; SAX, M. Graph-based security models for object-oriented software systems. In: **Proceedings of the International Conference on Software Engineering**. [S.l.: s.n.], 2022.

HAMAD, R. H. Automation testing and monitoring lab on the cloud for iot smart fleet system (atml and sfs). In: **Proceedings of the 2018 International Conference**. [s.n.], 2018. Acesso em: 22 abr. 2025. Disponível em: <https://dl.acm.org/doi/10.1145/3234698.3234740>.

HAMADAQA, E.; MARS, A.; ADI, W. **Physical security for fleet management systems**Journal of Cybersecurity and Privacy, 2019, v. 4, n. 1, p. 1–15. Acesso em: 17 maio 2025.

JúNIOR, H. **A automação de testes desempenha um papel crítico no desenvolvimento de software moderno**. 2025. Artigo da DBCCompany.

KALUVAKURI, V.; KHAMBAM, S. **Securing Telematics Data in Fleet Management: Integrating IAM with ML Models for Data Integrity in Cloud-Based Applications**. 2024. Acesso em: 02 maio 2025. Disponível em: <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4927214>.

MOROZ, S. **Aplicações de gestão de frotas e segurança da informação**. 2020. Acesso em: 18 abr. 2025. Disponível em: <https://doi.org/10.1016/j.future.2020.01.020>.

PARGAONKAR, Y. **Advancements in security testing methodologies for software quality assurance**International Journal of Software Engineering and Knowledge Engineering, 2023, v. 33, n. 5,

Security Leaders; KASPERSKY. **Pesquisa sobre segurança cibernética no Brasil**. 2024. Acesso em: 05 abr. 2025. Disponível em: <https://www.securityleaders.com.br>.

TAKANEN, A. et al. **Fuzzing for software security testing and quality assurance**. Elsevier, 2018. Acesso em: 20 jun. 2025. Disponível em: <https://books.google.com/books?hl=en&lr=&id=tKN5DwAAQBAJ&oi=fnd&pg=PR15&dq=software+quality+and+security+and+test+automation+and+fleet+management&ots=dQgJq_Ve1e&sig=wx_IOIPR1skMPOP1tr7PBVjs5BE>.

TUSING, T.; BROOKS, J. **Best practices for API testing with Postman and Insomnia**. 2023. Acesso em: 14 mar. 2025. Disponível em: <https://postman.com/blog/api-testing-best-practices/>.