INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CAMPUS TERESINA CENTRAL
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ISRAEL BENVINDO PEREIRA

SEGURANÇA SHIFT-LEFT EM SISTEMAS DE SAÚDE DIGITAL: avaliação com ferramentas automatizadas em Python

TERESINA, PIAUÍ
2025

ISRAEL BENVINDO PEREIRA

SEGURANÇA SHIFT-LEFT EM SISTEMAS DE SAÚDE DIGITAL: avaliação com ferramentas automatizadas em Python

Trabalho de Conclusão de Curso (monografia) apresentado como exigência parcial para obtenção do diploma do Curso de Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Teresina Central.

Orientadora: Profa. M.a. Cláutenis Carvalho Viana

TERESINA, PIAUÍ
2025

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD

Pereira, Israel Benvindo

P455s Segurança sheft-left em sistemas de saúde digital : avaliação com ferramentas automatizadas em python / Israel Benvindo Pereira. - 2025.
50 f.: il. color.

Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Teresina Central, 2025.
Orientador : Prof Me. Cláutenis Carvalho Viana .

1. Segurança de aplicações. 2. Shift-left segurity. 3. LGPD. 4. Saúde digital. I.Título.

CDD - 004.21

Elaborado por Tanize Maria Sales CRB 3/1024

Israel Benvindo Pereira

SEGURANÇA SHIFT-LEFT EM SISTEMAS DE SAÚDE DIGITAL: avaliação com ferramentas automatizadas em Python

Trabalho de Conclusão de Curso (monografia) apresentado como exigência parcial para obtenção do diploma do Curso de Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Teresina Central.

Aprovada em 19 de Dezembro de 2025.

BANCA EXAMINADORA:

Profa. M.a. Cláutenis Carvalho Viana (Orientadora)
Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

Prof. M.e. Wilson de Oliveira Junior
Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

Profa. Dr.a. Nadia Mendes dos Santos
Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

RESUMO

Este projeto apresenta uma abordagem prática para a aplicação de estratégias de segurança *Shift-Left* no contexto da saúde digital, com ênfase em sistemas de prontuário eletrônico. A pesquisa apresenta a fundamentação teórica sobre segurança de aplicações e demonstra a importância do desenvolvimento seguro desde as fases iniciais do ciclo de vida do software. Para endereçar esta questão, foi desenvolvida uma API REST em Python simulando um módulo de prontuários eletrônicos, na qual foram inseridas vulnerabilidades intencionais (Injeção de SQL, BOLA/IDOR e Autenticação Quebrada) para fins de estudo. A eficácia das estratégias de *Shift-Left* foi avaliada através da aplicação de ferramentas de Análise Estática de Segurança de Aplicações (SAST), como o **Bandit**, e de Análise Dinâmica (DAST), por meio de scripts desenvolvidos com **Pytest**. Adicionalmente, foi implementado um pipeline de integração contínua (CI/CD) com **GitHub Actions** que automatizou a execução desses testes, validando a capacidade da abordagem em detectar e barrar código inseguro de forma proativa. Os resultados obtidos demonstram que a antecipação das práticas de segurança contribui para o desenvolvimento de aplicações mais robustas e resilientes. O trabalho estabelece uma base metodológica que alinha as práticas técnicas de desenvolvimento seguro com as exigências da Lei Geral de Proteção de Dados (LGPD) e da Resolução CFM nº 2.314/2022, promovendo a integração entre a engenharia de software e a conformidade regulatória no setor da saúde digital.

Palavras-chave: segurança de aplicações; Shift-Left Security; LGPD; saúde digital; DevSecOps; Python.

ABSTRACT

This project presents a practical approach to applying *Shift-Left* security strategies in the context of digital health, with an emphasis on electronic health record systems. The research provides the theoretical foundation for application security and demonstrates the importance of secure development from the initial stages of the software development lifecycle. To address this issue, a REST API simulating an electronic health records module was developed in Python, into which intentional vulnerabilities (SQL Injection, BOLA/IDOR, and Broken Authentication) were inserted for study purposes. The effectiveness of the *Shift-Left* strategies was evaluated through the application of Static Application Security Testing (SAST) tools, such as **Bandit**, and Dynamic Application Security Testing (DAST), by means of scripts developed with **Pytest**. Additionally, a continuous integration and continuous delivery (CI/CD) pipeline was implemented using **GitHub Actions**, which automated the execution of these tests, validating the approach's ability to proactively detect and block insecure code. The results show that anticipating security practices contributes to the development of more robust and resilient applications. This work establishes a methodological foundation that aligns technical secure development practices with the requirements of the Brazilian General Data Protection Law (LGPD) and Federal Medical Council (CFM) Resolution No. 2,314/2022, promoting the integration between software engineering and regulatory compliance in the digital health sector.

Keywords: application security; Shift-Left Security; LGPD; digital health; DevSecOps; Python.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface (Interface de Programação de Aplicações)
AST	Application Security Testing (Análise de Segurança de Aplicações)
BOLA	Broken Object Level Authorization (Quebra de Autorização em Nível de Objeto)
BOLA/IDOR	Broken Object Level Authorization / Insecure Direct Object References
CFM	Conselho Federal de Medicina
CI	Continuous Integration (Integração Contínua)
CI/CD	Continuous Integration / Continuous Delivery (Integração Contínua / Entrega Contínua)
CRUD	Create, Read, Update, Delete
CVE	Common Vulnerabilities and Exposures
DAST	Dynamic Application Security Testing (Análise Dinâmica de Segurança de Aplicações)
DevSecOps	Development, Security and Operations
DoS	Denial of Service (Negação de Serviço)
HTTP	Hypertext Transfer Protocol
IaC	Infrastructure as Code (Infraestrutura como Código)
IDOR	Insecure Direct Object References
JSON	JavaScript Object Notation
JWT	JSON Web Token
LGPD	Lei Geral de Proteção de Dados Pessoais
OWASP	Open Worldwide Application Security Project
PE	Prontuário Eletrônico
QA	Quality Assurance (Garantia da Qualidade)

REST	Representational State Transfer
SAST	Static Application Security Testing (Análise Estática de Segurança de Aplicações)
SCA	Software Composition Analysis (Análise de Composição de Software)
SDLC	Software Development Life Cycle (Ciclo de Vida do Desenvolvimento de Software)
SQL	Structured Query Language
UI	User Interface (Interface do Usuário)
URL	Uniform Resource Locator
WAF	Web Application Firewall
ZAP	Zed Attack Proxy (OWASP ZAP)

LISTA DE SÍMBOLOS

–	Travessão (hífen longo)
º	Indicador ordinal
•	Marcador de item em listas
=	Operador de atribuição/igualdade
*	Asterisco (coringa)

SUMÁRIO

1 INTRODUÇÃO

A crescente complexidade dos sistemas de software, aliada à intensificação dos ataques cibernéticos, tem evidenciado as limitações das abordagens tradicionais de segurança, historicamente concentradas nas fases finais do ciclo de vida do desenvolvimento de software (SDLC). Essas abordagens reativas, como apontam Malik e Prashasti (2025), concentram-se na detecção de vulnerabilidades apenas nas etapas de teste ou implantação, estratégia que tem se mostrado insuficiente diante dos desafios contemporâneos. Nesse contexto, o conceito de *Shift-Left Security* surge como uma resposta estratégica, propondo a antecipação das práticas de segurança para as etapas iniciais do desenvolvimento, como o design e a codificação. Essa mudança de paradigma possibilita a identificação precoce de vulnerabilidades, a redução de custos com retrabalho e maior eficiência na entrega de software robusto e em conformidade com requisitos regulatórios.

A adoção do *Shift-Left Security* torna-se especialmente relevante em áreas críticas como a saúde, onde a digitalização avança rapidamente por meio da telemedicina e dos prontuários eletrônicos (PE). Embora essas tecnologias promovam maior acessibilidade e eficiência nos atendimentos, a proteção de dados emerge como uma preocupação central. Segundo Mocydlarz-Adamcewicz et al. (2023), a implementação de medidas robustas de segurança é indispensável nesse cenário, uma vez que a expansão tecnológica amplia significativamente a superfície de ataque, expondo dados sensíveis a riscos elevados. Falhas de segurança podem gerar consequências severas, como violações de privacidade, danos reputacionais e sanções legais.

No contexto brasileiro, a Resolução CFM nº 2.314/2022 (MEDICINA, 2022) estabelece diretrizes para a prática da telemedicina, exigindo que os dados dos pacientes sejam preservados conforme normas legais e éticas, garantindo integridade, confidencialidade e sigilo profissional. Adicionalmente, a normativa reforça a necessidade de protocolos de segurança alinhados aos princípios da Lei Geral de Proteção de Dados Pessoais (LGPD) (BRASIL, 2018), Lei nº 13.709, de 14 de agosto de 2018. Tais exigências regulatórias evidenciam a indispensabilidade de soluções técnicas eficazes e proativas para garantir tanto a conformidade legal quanto a proteção integral das informações clínicas.

Diante desse cenário, esta pesquisa propõe a avaliação de estratégias de *Shift-Left Security* por meio da construção e análise de uma aplicação de laboratório. Foi desenvolvida uma API REST em Python com vulnerabilidades intencionais, simulando um módulo de prontuários eletrônicos, sobre a qual foram aplicadas ferramentas de segurança estática (SAST), dinâmica (DAST) e um pipeline de integração contínua (CI/CD). Essa abordagem permite demonstrar, de forma prática, como a antecipação

de falhas contribui para o fortalecimento da segurança em sistemas de telemedicina e para a promoção da conformidade regulatória.

A concretização deste estudo envolveu a elaboração de um ambiente experimental controlado, no qual vulnerabilidades como Injeção de SQL, Quebra de Autorização em Nível de Objeto (BOLA/IDOR) e Autenticação Quebrada foram inseridas e, posteriormente, mitigadas. A execução prática possibilitou a validação da abordagem metodológica adotada e forneceu dados empíricos que fundamentam as conclusões sobre a eficácia do *Shift-Left Security* no contexto da segurança de aplicações para a saúde digital.

2 OBJETIVOS

2.1 OBJETIVO GERAL

Implementar e avaliar estratégias de *Shift-Left Security*, por meio da aplicação de varreduras automatizadas e da análise de código em Python, no desenvolvimento de um módulo de prontuários eletrônicos em ambiente de laboratório, no contexto da telemedicina.

2.2 OBJETIVOS ESPECÍFICOS

- **Investigar** os fundamentos do *Shift-Left Security* e suas implicações para a detecção proativa de vulnerabilidades em sistemas de telemedicina, considerando as diretrizes de segurança e os requisitos de conformidade legal estabelecidos pela LGPD e pela Resolução CFM.
- **Implementar** uma metodologia para a integração de varreduras de segurança automatizadas (SAST e DAST) em um pipeline de integração contínua (CI/CD) aplicado ao desenvolvimento de software de prontuários eletrônicos.
- **Avaliar** a eficácia da metodologia proposta na mitigação de vulnerabilidades de segurança (SQL Injection, BOLA/IDOR e Autenticação Quebrada) em uma API REST de laboratório, por meio da demonstração de ataques, da aplicação de contramedidas e da validação das correções.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta o arcabouço teórico que fundamenta a presente pesquisa, abordando inicialmente a evolução da segurança no desenvolvimento de software e o paradigma do *Shift-Left Security*, com seus princípios, benefícios e desafios de implementação. Em seguida, discute-se o conceito de vulnerabilidades em sistemas computacionais, com ênfase nos riscos associados a aplicações no contexto da saúde digital. O capítulo também descreve as ferramentas de *Application Security Testing* (AST), utilizadas para a detecção de falhas de segurança, e finaliza com a análise do contexto regulatório e ético da saúde digital no Brasil, destacando a intersecção entre as exigências legais e as práticas de segurança técnica adotadas neste trabalho.

3.1 SEGURANÇA NO CICLO DE VIDA DO DESENVOLVIMENTO DE SOFTWARE (SDLC)

O Ciclo de Vida do Desenvolvimento de Software (SDLC, do inglês *Software Development Life Cycle*) consiste em uma estrutura fundamental utilizada na indústria para conceber, desenvolver, testar e entregar produtos de software com qualidade, confiabilidade e dentro de prazos estabelecidos (PARGAONKAR, 2023). Entre os modelos clássicos de SDLC, destaca-se a metodologia Cascata (*Waterfall*), caracterizada por uma abordagem sequencial, na qual cada fase, desde a análise de requisitos até a manutenção, deve ser integralmente concluída antes do avanço para a etapa seguinte. Essa linearidade, contudo, limita a capacidade de adaptação e de revisões contínuas, especialmente no que se refere à segurança, tradicionalmente incorporada apenas nas fases finais do processo.

Com o avanço das demandas tecnológicas, a rigidez dos processos fortemente orientados à documentação passou a ser percebida como um fator limitante para desenvolvedores e organizações que não dispunham de recursos para modelos pesados de produção de software (SOARES, 2004). Nesse contexto, emergiram metodologias iterativas e ágeis, como o Scrum e o DevOps, que introduziram maior flexibilidade, entregas contínuas e a possibilidade de integração de testes ao longo de todo o desenvolvimento. Essas abordagens ampliaram o reconhecimento de que a segurança não deve ser tratada como um componente isolado ou tardio, mas sim como um requisito transversal a todas as etapas do ciclo de vida.

É nesse cenário que se consolida o conceito de *Shift-Left Security*, o qual propõe a antecipação sistemática das práticas de segurança para as fases iniciais do SDLC, como o design e a codificação. Conforme destacado por Malik e Prashasti (2025), a abordagem tradicional reativa, centrada em testes e validações ao final

do ciclo, mostrou-se insuficiente diante da crescente complexidade dos sistemas modernos e da sofisticação dos ataques cibernéticos. Ao incorporar a segurança desde as primeiras etapas, a filosofia *Shift-Left* favorece a identificação precoce de vulnerabilidades, reduz custos com retrabalho, mitiga riscos operacionais e fortalece a cultura de desenvolvimento seguro, sendo este o princípio central explorado no presente trabalho.

3.2 SHIFT-LEFT SECURITY

A filosofia *Shift-Left* teve sua origem em 2001, quando Larry Smith propôs uma integração mais profunda e antecipada entre o desenvolvimento de software e a garantia da qualidade (QA, do inglês *Quality Assurance*). Em seu artigo publicado no *Dr. Dobb's Journal*, Smith cunhou o termo *Shift-Left Testing* para descrever a prática de antecipar os testes no cronograma do projeto, promovendo a colaboração entre desenvolvedores e profissionais de QA desde as fases iniciais do processo (SMITH, 2001). Com o passar do tempo, essa filosofia foi expandida para outras áreas do ciclo de desenvolvimento, culminando na abordagem denominada *Shift-Left Security*, que aplica o mesmo princípio de antecipação às práticas de segurança da informação.

3.2.1 Conceito e princípios

A abordagem *Shift-Left Security* representa uma estratégia proativa que propõe a integração sistemática de práticas de segurança desde as fases iniciais do ciclo de vida do desenvolvimento de software (SDLC). Ao deslocar a segurança para as etapas iniciais do processo, busca-se antecipar a identificação e a mitigação de riscos, incorporando medidas preventivas já nas fases de planejamento, design e codificação (VORUGANTI, 2021). Diferentemente das abordagens tradicionais, que tratam a segurança de forma reativa e tardia, essa metodologia visa reduzir falhas críticas, aumentar a resiliência das aplicações e acelerar a entrega de software de forma segura.

Essa mudança de paradigma é sustentada por princípios fundamentais como a **segurança desde o design** (*security by design*), a **automação contínua de processos**, a **colaboração entre equipes** e o **feedback rápido e integrado**. A aplicação prática desses princípios, explorada neste trabalho, ocorre em diferentes etapas do SDLC, conforme descrito a seguir:

- **Design:** Definição de requisitos de segurança e realização da Modelagem de Ameaças (*Threat Modeling*), com o objetivo de identificar riscos ainda antes do início da codificação.

- **Codificação:** Aplicação de boas práticas de desenvolvimento seguro e utilização de ferramentas de Análise Estática de Código (SAST), como o Bandit, fornecendo *feedback* imediato ao desenvolvedor.
- **Testes:** Integração de testes de segurança automatizados em um pipeline de integração contínua (CI/CD), utilizando ferramentas de Análise Dinâmica de Aplicações (DAST) para validação da aplicação em execução.
- **Implantação e Operação:** Monitoramento contínuo e validação de infraestrutura como código (*Infrastructure as Code – IaC*), assegurando que os mecanismos de segurança permaneçam efetivos após a implantação.

3.2.2 Benefícios

A antecipação das práticas de segurança ao longo do SDLC, conforme preconiza a abordagem *Shift-Left*, proporciona uma série de benefícios relevantes (RANI et al., 2023):

- **Redução de custos:** A correção de vulnerabilidades nas fases iniciais do desenvolvimento apresenta custo significativamente inferior quando comparada à correção em ambiente de produção.
- **Detecção precoce:** Permite identificar e mitigar falhas antes que se tornem críticas, reduzindo a janela de exposição a ataques.
- **Eficiência operacional:** Diminui o retrabalho e acelera a entrega de software, uma vez que a segurança é validada continuamente por meio da automação.
- **Conformidade regulatória:** Facilita a adequação a normas como a Lei Geral de Proteção de Dados (LGPD) desde o início do projeto, evitando que a conformidade seja tratada apenas como uma etapa final.
- **Melhoria na qualidade do código:** Incentiva a adoção de padrões rigorosos de codificação segura, resultando em aplicações mais robustas e resilientes.

3.2.3 Desafios e implementação

Apesar das vantagens apresentadas, a adoção do *Shift-Left Security* impõe desafios organizacionais e técnicos relevantes (RANI et al., 2023):

- **Mudança cultural:** Um dos principais desafios consiste na consolidação da mentalidade de que a segurança é uma responsabilidade compartilhada, e não exclusiva da equipe de segurança. Para isso, os desenvolvedores devem ser capacitados a projetar e implementar código seguro.

- **Integração de ferramentas:** A seleção e a integração de ferramentas SAST e DAST aos pipelines de CI/CD podem apresentar complexidades técnicas, como observado no processo de configuração do pipeline desenvolvido neste trabalho.
- **Capacitação técnica:** Torna-se necessário investir continuamente em treinamentos sobre desenvolvimento seguro, novas ameaças e utilização adequada das ferramentas de segurança.
- **Gerenciamento de falsos positivos:** Ferramentas automatizadas podem gerar alertas que não correspondem a vulnerabilidades reais. Assim, faz-se necessário estabelecer processos de triagem e ajuste das ferramentas, como discutido no Capítulo 5 a respeito da configuração de severidade do Bandit.

A superação desses desafios exige uma atuação colaborativa entre as equipes de desenvolvimento, operações e segurança, caracterizando a abordagem *DevSecOps*, bem como investimentos contínuos em cultura organizacional, capacitação técnica e maturidade dos processos.

3.3 VULNERABILIDADES EM SISTEMAS DE SOFTWARE

No contexto da segurança de software, vulnerabilidades correspondem a falhas, fraquezas ou erros de implementação em um sistema que podem ser explorados por agentes maliciosos para realizar ações prejudiciais, tais como expor ou alterar informações sensíveis, comprometer a integridade de sistemas ou assumir o controle de aplicações (DOWD; MCDONALD; SCHUH, 2006). Em aplicações voltadas à área da saúde, como prontuários eletrônicos e plataformas de telemedicina, essas vulnerabilidades assumem caráter ainda mais crítico, em razão da alta sensibilidade dos dados tratados e das implicações legais e éticas associadas ao seu uso indevido.

Uma das principais referências para a identificação e categorização de vulnerabilidades em aplicações é a lista **OWASP Top 10** (OWASP Foundation, 2021), mantida pela *Open Worldwide Application Security Project*. Essa classificação apresenta as falhas mais recorrentes e críticas observadas em aplicações modernas. Para o escopo deste trabalho, foram selecionadas e analisadas três categorias de risco consideradas altamente relevantes para sistemas de saúde digital, as quais foram propositalmente implementadas e posteriormente mitigadas na aplicação de laboratório desenvolvida:

- **A01:2021 – Quebra de Controle de Acesso (*Broken Access Control*)**: Refere-se a falhas que permitem que usuários acessem dados ou funcionalidades para os quais não possuem permissão adequada. Essa categoria engloba a vulnerabilidade do tipo **BOLA/IDOR**, explorada neste estudo por meio da possibilidade de acesso indevido ao prontuário de outro paciente mediante a simples manipulação de identificadores em requisições HTTP.

- **A03:2021 – Injeção (*Injection*)**: Ocorre quando dados não confiáveis são enviados a um interpretador como parte de um comando ou consulta, possibilitando a execução de instruções não previstas. A vulnerabilidade de **Injeção de SQL** presente no endpoint de autenticação do projeto constitui um exemplo clássico dessa categoria, permitindo o *bypass* dos mecanismos de login.
- **A07:2021 – Falhas de Identificação e Autenticação (*Identification and Authentication Failures*)**: Relaciona-se a implementações incorretas ou fragilizadas de mecanismos de autenticação, como o gerenciamento inadequado de sessões e chaves criptográficas. No presente trabalho, essa categoria foi representada pela vulnerabilidade de **Autenticação Quebrada**, ocasionada pela exposição da chave secreta utilizada na assinatura de tokens JWT.

Segundo Ewoh e Vartiainen (2024), a complexidade crescente dos ambientes digitais na área da saúde, aliada ao avanço acelerado da digitalização, expõe os sistemas a múltiplos vetores de ataque. Entre os anos de 2009 e 2021, mais de 4.400 violações de dados no setor de saúde foram registradas apenas nos Estados Unidos, comprometendo mais de 314 milhões de registros, o que evidencia a magnitude dos riscos envolvidos.

Ainda de acordo com os autores, fatores como a deficiência em educação em segurança cibernética, a escassez de profissionais especializados e as lacunas existentes entre tecnologia, processos e comportamento humano contribuem significativamente para a fragilidade dos sistemas. Esses elementos reforçam a importância de abordagens proativas, como o *Shift-Left Security*, que promovem a detecção precoce e a mitigação preventiva dessas falhas desde as etapas iniciais do desenvolvimento, conforme demonstrado na parte prática deste trabalho.

3.4 FERRAMENTAS DE ANÁLISE DE SEGURANÇA DE APLICAÇÕES (AST)

A Análise de Segurança de Aplicações (AST, do inglês *Application Security Testing*) refere-se ao conjunto de técnicas, ferramentas e processos utilizados para identificar vulnerabilidades e falhas de segurança em aplicações de software. Essas ferramentas constituem componentes essenciais da estratégia *Shift-Left Security*, pois permitem a automatização da detecção de falhas em diferentes estágios do ciclo de vida do desenvolvimento. No presente trabalho, foram empregadas as duas principais abordagens de AST: a Análise Estática (SAST) e a Análise Dinâmica (DAST).

3.4.1 Análise estática de segurança de aplicações (SAST)

A Análise Estática de Segurança de Aplicações (SAST, *Static Application Security Testing*) consiste em uma abordagem de testes que examina o código-fonte,

o *bytecode* ou o código binário da aplicação sem a necessidade de sua execução. Trata-se de uma técnica classificada como *white-box testing*, pois analisa a estrutura interna do software, permitindo a identificação de vulnerabilidades ainda nas fases iniciais do SDLC. No contexto deste projeto, a ferramenta **Bandit** foi utilizada para a realização da análise SAST no código-fonte desenvolvido em Python (HORVATH; ZUMERLE; GARDNER, 2020).

Ao operar de forma estática, o SAST possibilita a detecção de falhas como injeção de comandos, uso de funções inseguras, tratamento inadequado de exceções e a exposição de segredos no código (*hard-coded secrets*). Essa abordagem favorece a correção precoce das vulnerabilidades, antes mesmo da aplicação ser disponibilizada em ambiente de execução. No presente trabalho, a eficácia da análise estática foi evidenciada na detecção das vulnerabilidades de Injeção de SQL e de Autenticação Quebrada, orientando a implementação das respectivas contramedidas.

3.4.2 Análise dinâmica de segurança de aplicações (DAST)

A Análise Dinâmica de Segurança de Aplicações (DAST, *Dynamic Application Security Testing*) realiza a avaliação da aplicação durante o seu estado de execução, geralmente nas fases de teste ou operação. Essa abordagem simula ataques contra os *endpoints* expostos da aplicação, como APIs e sistemas web, com o objetivo de observar seu comportamento e identificar vulnerabilidades efetivamente exploráveis (HORVATH; ZUMERLE; GARDNER, 2020).

Por se tratar de uma técnica de *black-box testing*, o DAST avalia a aplicação sob a perspectiva de um agente externo, sem acesso ao código-fonte. No presente trabalho, a análise dinâmica foi implementada por meio de **scripts customizados em Python**, utilizando as bibliotecas **Pytest** e **Requests** para automatizar a simulação de ataques e validar a eficácia dos mecanismos de segurança implementados.

3.4.3 Comparativo: SAST vs. DAST

A análise comparativa evidencia que as abordagens SAST e DAST possuem naturezas complementares, sendo ambas fundamentais em uma estratégia de segurança madura. A Tabela 1 resume as principais diferenças entre as duas metodologias.

Tabela 1 – Comparativo entre SAST e DAST

Característica	**SAST**	**DAST**
Tipo de análise	Estática (análise do código-fonte)	Dinâmica (análise da aplicação em execução)
Abordagem	*White-box testing*	*Black-box testing*
Momento de aplicação	Fases iniciais do desenvolvimento (codificação)	Fases de teste e operação
Cobertura	Todo o código analisado, incluindo caminhos não executados	Somente a superfície exposta da aplicação
Exemplos de ferramentas	Bandit, Safety, Semgrep	Scripts customizados (Requests, Pytest), OWASP ZAP, Burp Suite

Fonte: Elaborado pelo autor (2025).

Enquanto o SAST, exemplificado neste trabalho pelo Bandit, possibilita a identificação precoce de vulnerabilidades diretamente no código-fonte, o DAST, implementado através de scripts Pytest, permite a validação prática dessas falhas a partir do comportamento da aplicação em execução. A adoção conjunta dessas metodologias em um pipeline de integração contínua potencializa o nível de segurança da aplicação, ao combinar prevenção precoce com a validação dinâmica de ataques reais, conforme discutido por Dencheva (2022).

3.5 CONTEXTO REGULATÓRIO E ÉTICO NA SAÚDE DIGITAL

A transformação digital no setor da saúde impôs novos desafios relacionados à privacidade, confidencialidade e segurança dos dados sensíveis dos pacientes. Nesse cenário, o cumprimento das normas legais e éticas torna-se um pilar essencial para garantir a legitimidade, a confiança e a sustentabilidade dos sistemas de saúde digital, especialmente em contextos como a telemedicina e o uso de prontuários eletrônicos.

No Brasil, a vigência da Lei Geral de Proteção de Dados Pessoais (LGPD) estabeleceu um arcabouço legal que visa garantir a segurança dos dados, tornando as organizações responsáveis pela coleta, manipulação e tratamento de informações pessoais, sob pena de severas sanções em caso de descumprimento (SILVA; SANTOS, 2025). A aplicação de práticas de segurança robustas, como as propostas pela filosofia *Shift-Left*, é, portanto, não apenas uma boa prática técnica, mas um requisito fundamental para a conformidade legal.

3.5.1 Lei geral de proteção de dados pessoais (LGPD)

A Lei nº 13.709/2018, conhecida como Lei Geral de Proteção de Dados Pessoais (LGPD), estabelece diretrizes fundamentais para o tratamento de dados pessoais

no Brasil, aplicando-se de forma rigorosa ao setor da saúde. A LGPD reconhece os dados de saúde como **dados pessoais sensíveis**, conferindo-lhes um grau adicional de proteção devido ao seu potencial discriminatório e à sua relevância para a dignidade da pessoa humana (BRASIL, 2018).

O Art. 6º da lei elenca uma série de princípios que devem orientar todo o tratamento de dados, dos quais se destacam:

- **Finalidade:** os dados devem ser tratados para propósitos legítimos, específicos e explícitos, informados previamente ao titular.
- **Adequação:** o tratamento deve ser compatível com as finalidades informadas ao titular.
- **Necessidade:** a coleta deve se restringir ao mínimo necessário para a realização das finalidades.
- **Livre acesso:** deve-se garantir aos titulares consulta facilitada sobre a forma e a integralidade de seus dados.
- **Qualidade dos dados:** os dados devem ser exatos, claros, relevantes e atualizados.
- **Transparência:** as informações sobre o tratamento devem ser disponibilizadas de forma clara e acessível.
- **Segurança:** devem ser utilizadas medidas técnicas e administrativas aptas a proteger os dados de acessos não autorizados e de situações acidentais ou ilícitas.
- **Prevenção:** deve ocorrer a adoção de medidas para prevenir a ocorrência de danos em virtude do tratamento de dados.
- **Não discriminação:** impossibilidade de realização do tratamento para fins discriminatórios ilícitos ou abusivos.
- **Responsabilização e prestação de contas:** o agente deve demonstrar a adoção de medidas eficazes e capazes de comprovar o cumprimento das normas.

No contexto organizacional, a LGPD surgiu como resposta à crescente preocupação com o uso indevido de dados, como vazamentos de informações e falhas de segurança. Como observado por Rapôso et al. (2019), a digitalização ampliou a exposição de instituições a ataques, motivando investimentos em governança de dados. Para sistemas de saúde digital, o cumprimento desses princípios é mandatório.

Dentre os princípios listados, os de **Segurança** e **Prevenção** possuem uma conexão direta com a engenharia de software. As vulnerabilidades de **Injeção de SQL**

e **Autenticação Quebrada**, exploradas neste trabalho, representam falhas diretas na implementação de "medidas técnicas"de segurança. A vulnerabilidade de **BOLA/IDOR** viola tanto o princípio da Segurança quanto o do Livre Acesso. A metodologia *Shift-Left Security*, ao promover a Modelagem de Ameaças e os testes automatizados, atua como uma ferramenta para a aplicação prática do princípio da Prevenção, buscando evitar que tais falhas ocorram. O descumprimento dessas obrigações pode acarretar sanções administrativas, multas e danos reputacionais significativos às instituições de saúde.

3.5.2 Resolução CFM nº 2.314/2022

No âmbito específico da medicina, a Resolução CFM nº 2.314/2022, do Conselho Federal de Medicina (CFM), regulamenta a prática da telemedicina no Brasil (MEDICINA, 2022). Essa norma estabelece os requisitos técnicos e éticos que devem ser observados para garantir a segurança da informação, a confidencialidade e o sigilo profissional nas interações médicas mediadas por tecnologia.

Entre as diretrizes estabelecidas, destacam-se:

- **Utilização de plataformas seguras:** Exigência de uso de sistemas que assegurem a integridade, a autenticidade e a confidencialidade dos dados dos pacientes.
- **Adoção de medidas de segurança da informação:** Implementação de controles técnicos e administrativos alinhados com as boas práticas do setor.
- **Armazenamento seguro dos prontuários eletrônicos:** Obrigatoriedade de garantir que os registros estejam protegidos contra acessos indevidos ou vazamentos.
- **Consentimento livre e esclarecido:** Respeito à autorização expressa do paciente quanto à coleta e uso de seus dados em ambiente digital.

A Resolução reforça, portanto, o caráter ético da segurança digital, vinculando a qualidade técnica das ferramentas à proteção da relação médico-paciente. Falhas de segurança como as exploradas neste trabalho, que permitem **acessos indevidos** (BOLA/IDOR), comprometem a **autenticidade** (Autenticação Quebrada) e a **integridade** dos dados (Injeção de SQL), representam violações diretas a essas diretrizes, sublinhando a importância de metodologias de desenvolvimento seguro como o *Shift-Left*.

3.5.3 Lei nº 14.510/2022: marco legal da telessaúde

A Lei nº 14.510, de 27 de dezembro de 2022, estabelece as diretrizes para a prática da telessaúde no Brasil, autorizando de forma permanente a prestação remota de serviços em todas as profissões regulamentadas da área da saúde (Brasil, 2022). Esta legislação representa um marco para a consolidação da saúde digital no país, promovendo o acesso ampliado, especialmente em regiões de difícil cobertura presencial.

Dentre os princípios que a lei estabelece, o da **Confidencialidade dos dados** é o que possui a mais forte intersecção com a segurança de software. O Art. 4º, Inciso V, assegura "a confidencialidade dos dados, garantindo a privacidade das informações sensíveis do paciente".

Ao estabelecer a **Confidencialidade dos dados** como um de seus pilares, a Lei nº 14.510/2022 torna a implementação de medidas de segurança técnica uma obrigação legal. Vulnerabilidades como as demonstradas neste trabalho, que podem levar ao acesso não autorizado e vazamento de informações sensíveis, representam uma violação direta a este princípio, reforçando que a adoção de práticas de desenvolvimento seguro é um requisito indispensável para a prática legal da telessaúde no Brasil.

3.5.4 Intersecção entre legislação e segurança técnica

A análise conjunta da LGPD, da Resolução CFM e do Marco Legal da Telessaúde evidencia que a segurança da informação deixou de ser apenas um requisito técnico para se tornar uma obrigação jurídica no desenvolvimento de software para a saúde. É nesta intersecção que a filosofia *Shift-Left Security* demonstra seu valor estratégico.

A implementação de estratégias de *Shift-Left*, como a Modelagem de Ameaças na fase de design, reforça diretamente o princípio da **Prevenção** (Art. 6º, VIII da LGPD). Da mesma forma, o uso de ferramentas de AST, como o SAST e o DAST, ao longo do desenvolvimento, materializa a exigência por "medidas técnicas [...] aptas a proteger os dados"(Princípio da **Segurança**, Art. 6º, VII da LGPD) e as salvaguardas requeridas pela Resolução CFM.

Além disso, a automação dessas práticas em um pipeline de CI/CD, conforme demonstrado neste trabalho, permite a rastreabilidade e a documentação contínua do processo de segurança. Isso atende diretamente ao princípio da **Responsabilização e Prestação de Contas** (Art. 6º, X da LGPD), pois a organização passa a ter evidências concretas de que adotou medidas eficazes para o cumprimento das normas.

Dessa forma, a integração entre a governança jurídica e a engenharia de software segura é indispensável para a construção de soluções em saúde digital. A abordagem *Shift-Left* emerge, portanto, não apenas como uma metodologia para criar

software mais seguro, mas como um caminho prático para desenvolver aplicações que sejam tecnicamente robustas, legalmente conformes e eticamente responsáveis.

4 METODOLOGIA

A presente seção descreve os procedimentos e o delineamento metodológico adotados para o desenvolvimento e a avaliação das estratégias de *Shift-Left Security* no contexto de sistemas de telemedicina. Detalham-se a abordagem de pesquisa empregada, a construção do ambiente simulado, a seleção das ferramentas e as etapas de aplicação das varreduras automatizadas que permitiram a obtenção de resultados válidos e reprodutíveis para embasar as discussões e conclusões deste trabalho.

4.1 VISÃO GERAL

A metodologia adotada neste projeto contemplou o desenvolvimento de uma aplicação experimental no contexto da saúde digital, com foco na segurança desde as etapas iniciais do ciclo de desenvolvimento. A aplicação consistiu em uma API REST escrita em Python, simulando um módulo de prontuários eletrônicos com funcionalidades básicas de autenticação, autorização e manipulação de dados clínicos.

Essa aplicação foi construída com a inserção intencional de vulnerabilidades baseadas no OWASP API Security Top 10, servindo como um ambiente de laboratório para a aplicação de estratégias de *Shift-Left Security*. Foram utilizadas ferramentas de *Application Security Testing* (AST), com foco em análise estática (SAST) durante a fase de codificação e análise dinâmica (DAST) com a aplicação em execução.

O objetivo foi demonstrar, de forma prática, como a antecipação de preocupações com segurança pode ser incorporada desde as primeiras fases do desenvolvimento, contribuindo para a construção de aplicações mais seguras e alinhadas com as exigências legais e éticas no setor da saúde.

4.2 ARQUITETURA E TECNOLOGIAS UTILIZADAS

A escolha das tecnologias para a parte prática deste trabalho fundamentou-se em critérios de acessibilidade, compatibilidade com os objetivos do projeto e aderência ao cenário de desenvolvimento de aplicações em saúde digital. Foram adotadas ferramentas amplamente utilizadas na indústria e na comunidade acadêmica, que possibilitaram tanto a simulação realista de vulnerabilidades quanto a aplicação das estratégias de segurança preconizadas pelo paradigma do *Shift-Left Security*. A arquitetura da solução consistiu em uma API REST desenvolvida em Python, interagindo com um banco de dados relacional e submetida a um pipeline de testes automatizados.

- **Python 3.11:** Linguagem de programação escolhida por sua simplicidade, legibili-

dade e ampla adoção em aplicações web e automações de teste. No setor de saúde, destaca-se em projetos de ciência de dados clínicos e análise de dados, impulsionada pelo caráter *open-source* e por bibliotecas como *Pandas*, *NumPy* e *Scikit-learn* (FILHO, 2015).

- **FastAPI:** Framework web de alta performance em Python, utilizado para o desenvolvimento da API REST devido à sua tipagem de dados com Pydantic, geração automática de documentação interativa (Swagger UI) e suporte a injeção de dependências, que foi fundamental para a implementação da autenticação.
- **SQLite:** Banco de dados relacional incorporado à biblioteca padrão do Python. Sua utilização permitiu o armazenamento persistente de dados de usuários e prontuários de forma simples, sem a necessidade de infraestrutura complexa, ao mesmo tempo em que possibilitou a simulação realista de ataques como a Injeção de SQL.
- **Passlib e Python-JOSE:** Conjunto de bibliotecas de segurança utilizadas para implementar a autenticação. A **Passlib**, com o algoritmo **bcrypt**, foi usada para o armazenamento seguro de senhas (hashing), enquanto a **Python-JOSE** foi empregada para a criação e validação de tokens de acesso no padrão JSON Web Token (JWT).
- **Bandit:** Ferramenta de Análise Estática de Segurança de Aplicações (SAST) voltada para códigos em Python. O Bandit foi utilizado para analisar o código-fonte em busca de padrões inseguros, como chaves secretas "hard-coded", exemplificando a detecção de falhas na fase de codificação.
- **Pytest e Requests:** A biblioteca **Requests** foi utilizada para a construção de scripts de testes dinâmicos (DAST), simulando interações com a API REST e ataques às vulnerabilidades implementadas. O **Pytest** foi adotado como o framework de execução desses testes, permitindo a verificação automatizada através de asserções (‘asserts‘).
- **GitHub Actions:** Ferramenta de Integração e Entrega Contínua (CI/CD) nativa do GitHub. Foi utilizada para construir um pipeline de DevSecOps completo, automatizando a execução dos testes de SAST (com Bandit) e DAST (com Pytest) a cada alteração no código, além de enviar notificações por e-mail em caso de falhas de segurança.

4.2.1 Justificativa da seleção tecnológica

A seleção tecnológica esteve intrinsecamente alinhada aos objetivos deste projeto, que visaram demonstrar, de forma prática e acessível, como as estratégias de

Shift-Left Security podem ser aplicadas ao desenvolvimento de um módulo simulado de prontuários eletrônicos. A escolha foi pautada na necessidade de ferramentas de código aberto (*open-source*) que oferecessem flexibilidade para a simulação de um ambiente realista, robustez para a aplicação de varreduras automatizadas e compatibilidade para integração no ciclo de desenvolvimento seguro.

Dessa forma, o conjunto de tecnologias adotado não apenas viabilizou a prova de conceito do *Shift-Left*, mas também permitiu uma análise aprofundada da detecção, mitigação e automação da segurança em um contexto relevante para a saúde digital. As ferramentas escolhidas permitiram, em suma:

- A criação de um ambiente de laboratório realista e controlado;
- A inserção e exploração de vulnerabilidades baseadas no OWASP API Security Top 10;
- A aplicação de testes de segurança estáticos (SAST) e dinâmicos (DAST) em fases iniciais do ciclo de desenvolvimento;
- A demonstração da integração automatizada de práticas de segurança ao ciclo de vida do software através de um pipeline de CI/CD.

Portanto, o conjunto tecnológico garantiu que o componente prático fosse ao mesmo tempo instrutivo, replicável e compatível com os princípios de segurança exigidos por normativas legais e técnicas.

4.3 DESENVOLVIMENTO DA APLICAÇÃO DE LABORATÓRIO

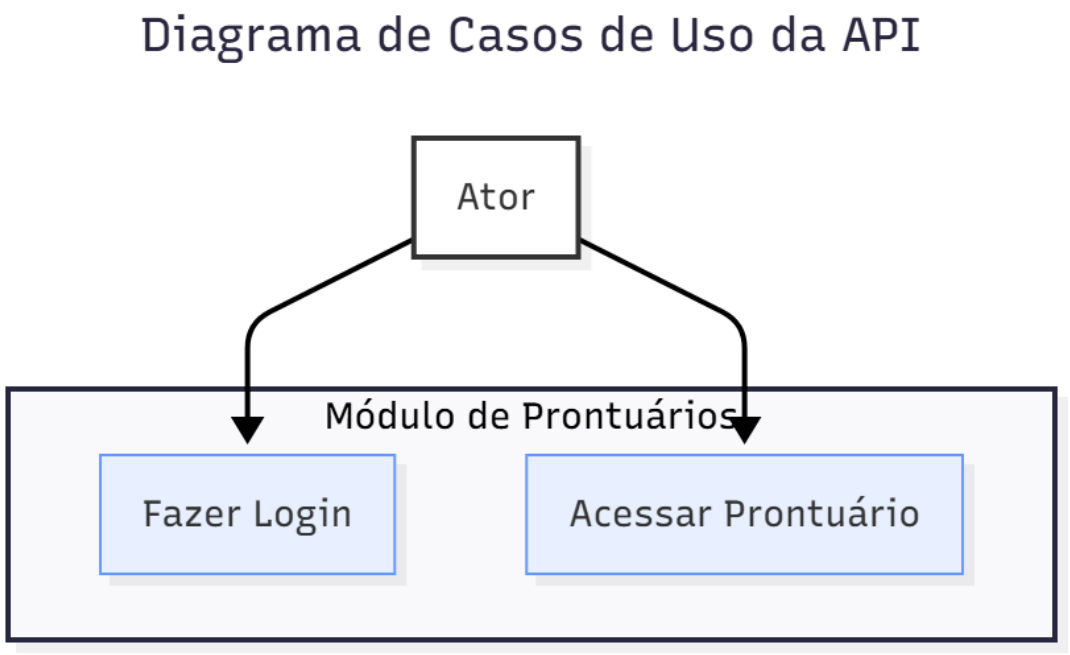
No contexto deste trabalho, foi desenvolvida uma aplicação de laboratório com o objetivo de reproduzir cenários realistas de vulnerabilidades em sistemas de saúde digital. Essa abordagem possibilitou a aplicação e avaliação prática das estratégias de segurança antecipadas (*Shift-Left Security*).

A aplicação consistiu na construção de uma API REST, desenvolvida em Python utilizando o framework FastAPI, que simulou um módulo de prontuários eletrônicos. Sendo um componente central em sistemas de telessaúde que lida com dados sensíveis, a API serviu como um objeto de estudo ideal para a análise de segurança. Suas funcionalidades abrangeram operações de leitura de dados, autenticação de usuários e um sistema de autorização baseado em tokens. A estrutura da aplicação foi projetada de forma modular para permitir a identificação clara dos pontos de segurança a serem explorados.

Para ilustrar as interações e funcionalidades da API, a **Figura 1** apresenta um diagrama de casos de uso simplificado, refletindo o escopo implementado neste trabalho. O diagrama detalha as interações de um **Ator** genérico com o sistema,

focando nas duas funcionalidades centrais que serviram de base para os estudos de segurança: o processo de autenticação (`Fazer Login`) e o acesso aos dados (`Acessar Prontuário`).

Figura 1 – Diagrama de Casos de Uso da API de Prontuários Eletrônicos



Fonte: Elaborado pelo autor (2025).

Com fins didáticos e para a avaliação prática das ferramentas AST, a aplicação foi construída com a inserção deliberada de um conjunto selecionado de vulnerabilidades descritas na lista OWASP API Security Top 10. As falhas implementadas e analisadas foram:

- **Injeção de SQL (*SQL Injection*)**: Um campo no endpoint de login foi deixado vulnerável à injeção de comandos SQL, permitindo o bypass do mecanismo de autenticação.
- **Quebra de Autorização em Nível de Objeto (*Broken Object Level Authorization* - BOLA/IDOR)**: O endpoint de acesso a prontuários foi implementado sem a verificação adequada de propriedade, permitindo que um usuário autenticado acessasse dados de outro usuário.
- **Autenticação Quebrada (*Broken Authentication*)**: O mecanismo de autenticação baseado em JWT foi implementado com uma chave secreta fraca e exposta no código-fonte ("hard-coded"), possibilitando a falsificação de tokens de acesso por um atacante.

A escolha dessas vulnerabilidades se baseou em sua alta frequência e impacto em aplicações do setor de saúde, conforme relatado por estudos e relatórios técnicos

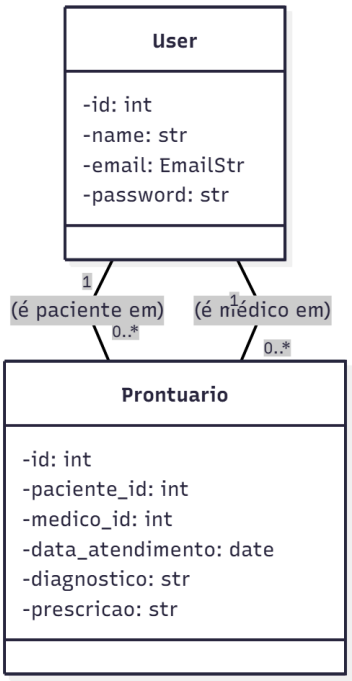
recentes (EWOH; VARTIAINEN, 2024). A abordagem de inserir falhas intencionais permitiu a validação prática das ferramentas de segurança empregadas, ilustrando os benefícios do pensamento seguro desde o início do ciclo de vida da aplicação.

O código-fonte completo do projeto foi mantido em um reposrito Git, permitindo o versionamento das correções e a implementação de um pipeline de integração contínua com as ferramentas de análise de segurança.

4.3.1 Modelo de dados da aplicação

Para detalhar a estrutura interna e as principais entidades de dados da API, o diagrama de classes da Figura 2 apresenta os modelos de dados implementados, seus atributos e os relacionamentos entre eles. Este modelo reflete a estrutura dos dados persistidos no banco de dados e utilizados para a validação e serialização das requisições e respostas da API através dos modelos Pydantic.

Figura 2 – Diagrama de Classes Simplificado dos Modelos de Dados da API



Fonte: Elaborado pelo autor (2025).

O modelo de dados define duas entidades centrais: **User** e **Prontuario**. A entidade `User` representa um usuário genérico do sistema, contendo informações básicas de identificação e credenciais. Neste escopo de trabalho, não foi implementada a distinção por papéis (como Médico e Paciente) em classes separadas, sendo a diferenciação gerenciada pela lógica de negócio e pelas relações no banco de dados.

A entidade `Prontuario` armazena os dados médicos e possui uma relação direta com dois usuários: o paciente (identificado por `paciente_id`) e o médico responsável (identificado por `medico_id`). Diferentemente do planejamento inicial, não

foram implementados métodos de negócio diretamente nas classes do modelo, seguindo a prática comum em APIs REST onde a lógica de negócio reside nos endpoints de serviço, que operam sobre esses modelos de dados puros.

4.4 APLICAÇÃO DAS ESTRATÉGIAS DE SEGURANÇA

A abordagem metodológica deste trabalho fundamentou-se na aplicação de estratégias alinhadas ao paradigma do *Shift-Left Security*, que visa antecipar as práticas de segurança para as fases iniciais do desenvolvimento de software. Dessa forma, foram incorporados mecanismos de identificação e mitigação de riscos em diferentes estágios do ciclo de vida da aplicação de laboratório.

Este processo foi dividido em três fases principais, que serão detalhadas nas subseções seguintes: a Modelagem de Ameaças, aplicada à fase de design; a Análise Estática (SAST), integrada à fase de codificação; e a Análise Dinâmica (DAST), executada com a aplicação em funcionamento para validar os controles de segurança.

4.4.1 Design seguro por modelagem de ameaças (*threat modeling*)

Conforme a filosofia *Shift-Left Security*, a primeira etapa de segurança foi aplicada ainda na fase de design, antes da implementação do código, através de um exercício de Modelagem de Ameaças (*Threat Modeling*). Trata-se de uma técnica de análise estrutural que permite identificar, classificar e planejar a mitigação de vulnerabilidades potenciais. Esta atividade possibilita a análise da arquitetura da aplicação, a identificação de ameaças e a definição de contramedidas técnicas apropriadas, servindo como um guia para um desenvolvimento mais seguro (DHILLON, 2011; FRYDMAN et al., 2014).

Para a realização da modelagem, foi adotado um processo baseado em quatro etapas: (i) identificação dos ativos críticos da aplicação; (ii) mapeamento dos agentes de ameaça; (iii) levantamento das vulnerabilidades mais relevantes para o contexto de uma API de saúde; e (iv) definição das contramedidas de segurança a serem aplicadas durante o desenvolvimento.

Com base nessa análise, a modelagem concentrou-se nas três classes deulnerabilidades que foram efetivamente implementadas e estudadas neste trabalho: **Injeção de SQL**, **Autenticação Quebrada** e **Quebra de Autorização em Nível de Objeto (BOLA/IDOR)**. A vulnerabilidade de injeção foi considerada devido ao seu potencial de comprometer a confidencialidade e a integridade da base de dados. A falha de autenticação foi analisada a partir do risco de falsificação de identidade por meio da forja de tokens JWT. Por fim, a vulnerabilidade BOLA/IDOR foi considerada em razão da possibilidade de acesso indevido a prontuários de terceiros.

As contramedidas prioritárias definidas nesta fase foram: o uso de consultas

parametrizadas, a gestão segura de chaves criptográficas através de variáveis de ambiente e a validação de propriedade (*ownership verification*) nos endpoints de acesso a dados.

A modelagem de ameaças, detalhada nas subseções seguintes, reforçou os princípios de segurança desde o design (*security by design*) e orientou diretamente as etapas de implementação e testes. Além disso, alinhou o desenvolvimento técnico às exigências de prevenção e proteção de dados previstas na Lei Geral de Proteção de Dados Pessoais (LGPD) e na Resolução CFM nº 2.314/2022.

4.4.1.1 Identificação de ativos e agentes de ameaça

Os ativos correspondem aos componentes do sistema que possuem valor e requerem proteção. Os agentes de ameaça são as entidades que podem tentar comprometer esses ativos. As Tabelas 2 e 3 apresentam os principais elementos identificados.

Tabela 2 – Principais ativos da aplicação

Ativo	**Categoria**	**Impacto em caso de violação**
Prontuário eletrônico	Dado sensível	Vazamento de dados de saúde, violando a LGPD e o sigilo médico.
Dados pessoais de usuários	Dado pessoal	Exposição indevida de dados cadastrais e risco de roubo de identidade.
Credenciais (hash de senha)	Credencial de autenticação	Quebra de senha por ataques offline (*offline cracking*) e sequestro de conta.
Token JWT	Credencial digital	Falsificação de identidade e acesso não autorizado a recursos protegidos.
Banco de dados SQLite	Infraestrutura	Vazamento massivo e centralizado de todas as informações da aplicação.
Endpoints da API	Superfície de ataque	Exploração de falhas na lógica da aplicação por meio de requisições maliciosas.

Fonte: Elaborado pelo autor (2025).

Tabela 3 – Atores do sistema e potenciais agentes de ameaça

Ator / Agente	**Descrição**
Usuário legítimo (Paciente ou Médico)	Ator autorizado que acessa a aplicação conforme seu perfil e permissões definidas pelo sistema.
Atacante externo	Agente não autenticado que explora vulnerabilidades expostas nos *endpoints* da API, como Injeção de SQL e falhas de autenticação.
Usuário autenticado mal-intencionado	Usuário legítimo que tenta exceder seus privilégios para acessar dados de terceiros, caracterizando ataques do tipo BOLA/IDOR.

Fonte: Elaborado pelo autor (2025).

4.4.1.2 Mapeamento de ameaças, vulnerabilidades e contramedidas

A Tabela 4, elaborada durante a fase de design, mapeia a relação entre as ameaças, as vulnerabilidades potenciais e as contramedidas planejadas, as quais foram posteriormente implementadas e validadas neste trabalho.

Tabela 4 – Mapeamento de Ameaças, Vulnerabilidades e Contramedidas

Ameaça	**Vulnerabilidade**	**Contramedida**
Acesso indevido a prontuários de outros pacientes	BOLA / IDOR	Verificação de propriedade (*ownership verification*) em todas as requisições de acesso aos dados.
Falsificação de identidade de usuário legítimo	Autenticação quebrada (JWT exposto)	Armazenamento seguro da chave secreta em variáveis de ambiente, isoladas do código-fonte.
Bypass do login e acesso irrestrito ao banco de dados	Injeção de SQL	Uso exclusivo de consultas parametrizadas (*prepared statements*) em todas as operações de banco de dados.

Fonte: Elaborado pelo autor (2025).

A análise de risco classificou todas as três ameaças como de **Alto Risco**, devido ao impacto Crítico da violação de dados de saúde, justificando a priorização de suas respectivas contramedidas. As etapas de testes descritas na sequência deste capítulo serviram para validar empiricamente a eficácia dessas contramedidas.

4.5 CICLO DE DETECÇÃO E MITIGAÇÃO DE VULNERABILIDADES

Após a fase de design, a metodologia prosseguiu com a implementação da API e a aplicação prática das estratégias de segurança. Para cada uma das vulnerabilidades identificadas na Modelagem de Ameaças, foi executado um ciclo completo de Shift-Left, consistindo em: (1) implementação da funcionalidade de forma intencionalmente vulnerável; (2) demonstração do ataque e suas implicações legais; (3) detecção da falha

com ferramentas SAST e/ou DAST; (4) implementação da contramedida de segurança; e (5) validação da correção através de testes de regressão.

As subseções seguintes detalham a aplicação deste ciclo para cada uma das vulnerabilidades estudadas.

4.5.1 Caso de estudo 1: injeção de sql (sql injection)

A primeira vulnerabilidade implementada foi a de Injeção de SQL no endpoint de autenticação. A falha, introduzida para fins didáticos, consistiu na construção de uma consulta SQL por meio da concatenação direta de dados fornecidos pelo usuário, uma prática insegura que abre margem para a manipulação da lógica da consulta. O trecho de código vulnerável no endpoint `/login` é apresentado na Listagem 4.1.

Listagem 4.1 – Código-fonte da vulnerabilidade de SQL Injection

```
1 query = f"SELECT * FROM users WHERE email = '{login_data.email}' AND
     password = '{login_data.password}';"
2
3 cursor = conn.cursor()
4 cursor.execute(query)
5 user_from_db = cursor.fetchone()
```

Fonte: Elaborado pelo autor (2025).

4.5.1.1 Demonstração de ataque e implicações

A exploração da vulnerabilidade foi demonstrada através do envio de um *payload* malicioso no campo de senha da requisição, conforme ilustra a Figura 3.

Figura 3 – Payload malicioso utilizado para o ataque de SQL Injection.

Fonte: Elaborado pelo autor (2025).

O *payload* `' OR '1'='1' --` altera a cláusula `WHERE` da consulta SQL para que ela sempre retorne um resultado verdadeiro, o que permitiu o *bypass* completo do mecanismo de autenticação. A Figura 4 evidencia o sucesso do ataque, onde o sistema concedeu acesso sem credenciais válidas, retornando o status HTTP `200 OK` e os dados do primeiro usuário da base de dados.

Figura 4 – Resultado bem-sucedido do ataque de bypass de autenticação.

Fonte: Elaborado pelo autor (2025).

Este acesso não autorizado representa uma violação direta de múltiplos dispositivos legais e regulatórios. A falha compromete frontalmente o **Princípio da Segurança**, detalhado no Art. 6º, Inciso VII, da LGPD, que exige:

> [...] utilização de medidas técnicas e administrativas aptas a proteger os dados pessoais de acessos não autorizados e de situações acidentais ou ilícitas de destruição, perda, alteração, comunicação ou difusão; (BRASIL, 2018, Art. 6º, VII).

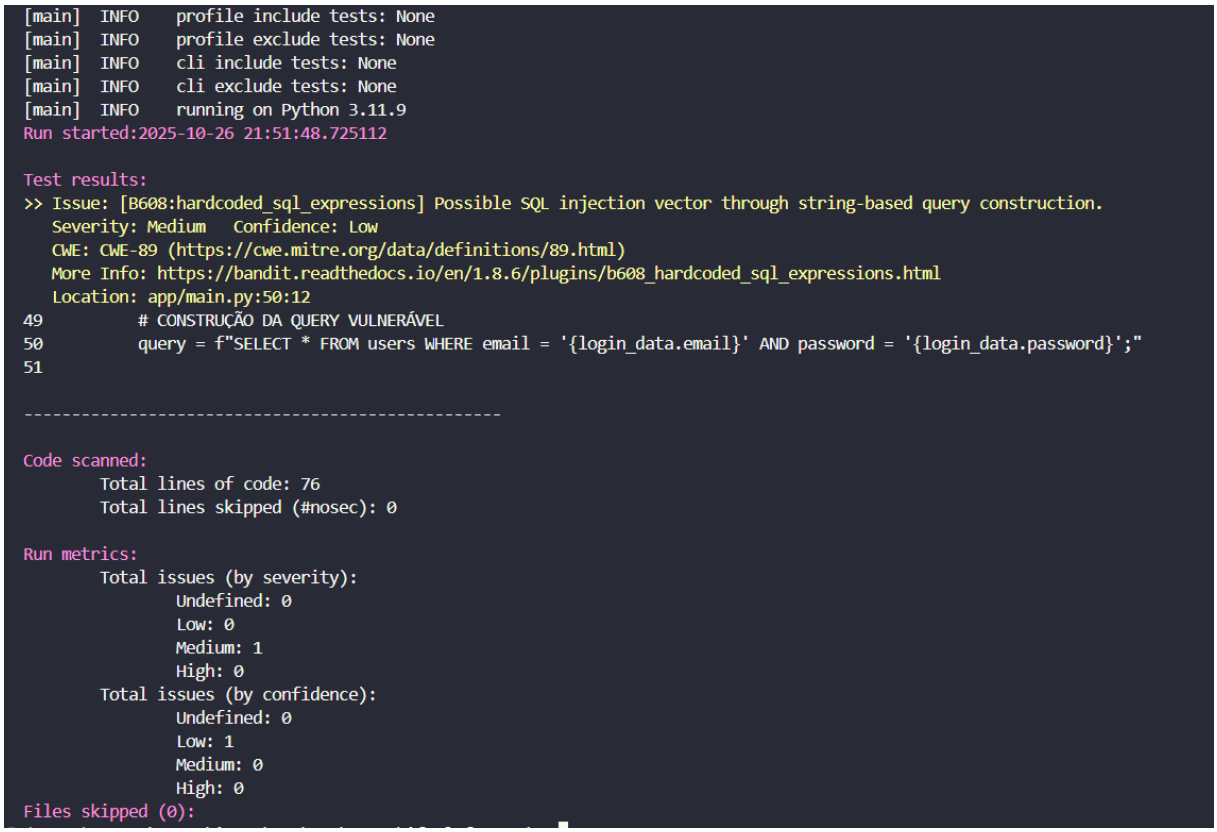
A ausência de uma proteção fundamental como o uso de consultas parametrizadas para prevenir Injeção de SQL constitui uma negligência clara na adoção das "medidas técnicas aptas"requeridas pela lei.

Adicionalmente, o acesso indevido viola o Art. 5º da Resolução CFM nº 2.314/2022, que determina que os sistemas devem garantir a **confidencialidade, a privacidade e o sigilo profissional**. A capacidade de um atacante acessar o sistema sem credenciais válidas invalida fundamentalmente essas garantias, contrariando a norma do conselho.

4.5.1.2 Detecção com sast e correção

Aplicando os princípios do Shift-Left, a detecção precoce da falha foi realizada utilizando a ferramenta de análise estática (SAST) **Bandit**. A Figura 5 exibe o relatório gerado pelo Bandit, que analisou o código-fonte e identificou com precisão a linha vulnerável, classificando-a como um risco de segurança sob o identificador `B608:hardcoded_sql_expressions`.

Figura 5 – Relatório da ferramenta SAST Bandit identificando a vulnerabilidade no código-fonte.



Fonte: Elaborado pelo autor (2025).

A contramedida implementada foi a refatoração da consulta para utilizar parâmetros (*prepared statements*), o que garante que os dados do usuário sejam tratados como valores literais e não como parte executável do comando SQL, conforme demonstrado na Listagem 4.2.

Listagem 4.2 – Código-fonte corrigido utilizando parâmetros de consulta

```
1 query = "SELECT * FROM users WHERE email = ? AND password = ?;"
2 params = (login_data.email, login_data.password)
3
4 cursor = conn.cursor()
5 cursor.execute(query, params)
6 user_from_db = cursor.fetchone()
```

Fonte: Elaborado pelo autor (2025).

Para validar a eficácia da correção, o mesmo ataque foi reexecutado. A Figura 6 demonstra a falha do ataque após a aplicação da contramedida, com a API retornando o erro `401 Unauthorized`, bloqueando efetivamente o acesso indevido.

Figura 6 – Resultado do ataque após a implementação da correção de segurança.

Fonte: Elaborado pelo autor (2025).

4.5.2 Caso de estudo 2: quebra de autorização em nível de objeto (bola/idor)

A segunda vulnerabilidade analisada foi a Quebra de Autorização em Nível de Objeto, frequentemente referida como BOLA ou IDOR (*Insecure Direct Object References*). A falha foi introduzida no endpoint `/prontuarios/{prontuario_id}`, onde a lógica inicial recuperava os dados do prontuário com base apenas no ID fornecido, sem realizar qualquer verificação de propriedade ou permissão.

Listagem 4.3 – Código-fonte da vulnerabilidade BOLA/IDOR

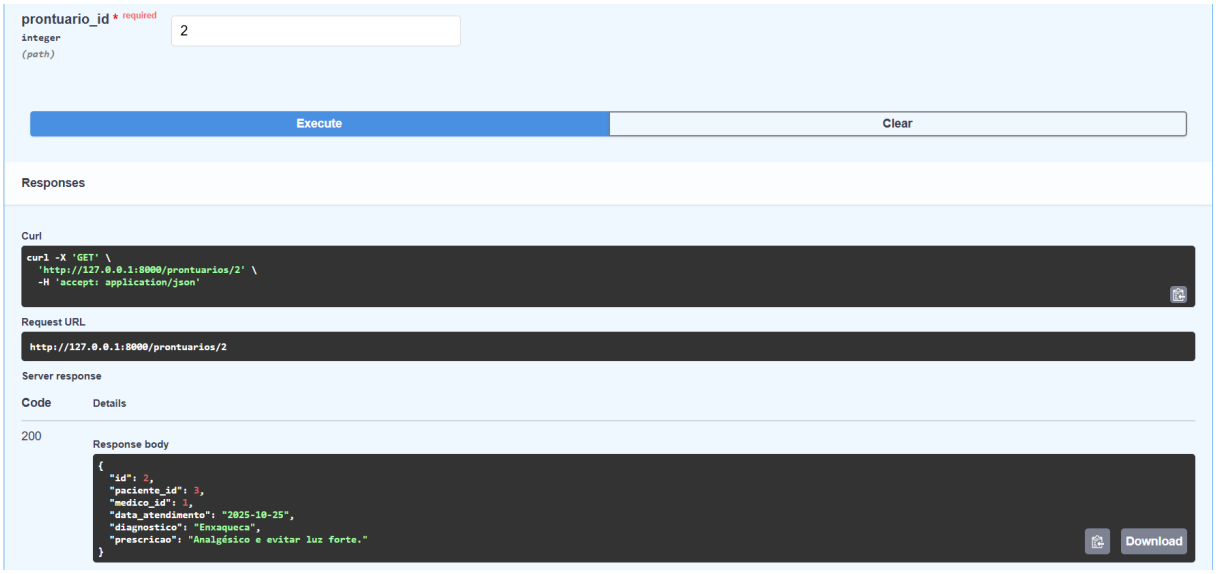
```
1 conn = get_db_connection()
2 prontuario = conn.execute("SELECT * FROM prontuarios WHERE id = ?;",
      (prontuario_id,)).fetchone()
3 conn.close()
4
5 if prontuario:
6     return Prontuario(**prontuario)
```

Fonte: Elaborado pelo autor (2025).

4.5.2.1 Demonstração de ataque e implicações

A exploração desta vulnerabilidade é trivial: um atacante, mesmo não autenticado, pode iterar valores numéricos para o `prontuario_id` na URL e acessar dados confidenciais de múltiplos pacientes. A Figura 7 demonstra o acesso indevido ao prontuário de `id=2`, que não pertence a um usuário logado.

Figura 7 – Sucesso do ataque BOLA/IDOR, exibindo dados de outro paciente.



Fonte: Elaborado pelo autor (2025).

Do ponto de vista legal, o acesso indevido a dados de terceiros por meio de uma falha BOLA/IDOR representa uma violação direta e grave a múltiplos dispositivos. Primeiramente, a vulnerabilidade contraria o **Princípio do Livre Acesso**, estabelecido no Art. 6º, Inciso IV, da LGPD, que garante exclusivamente ao titular dos dados:

> [...] a consulta facilitada e gratuita sobre a forma e a duração do tratamento, bem como sobre a integralidade de seus dados pessoais; (BRASIL, 2018, Art. 6º, IV).

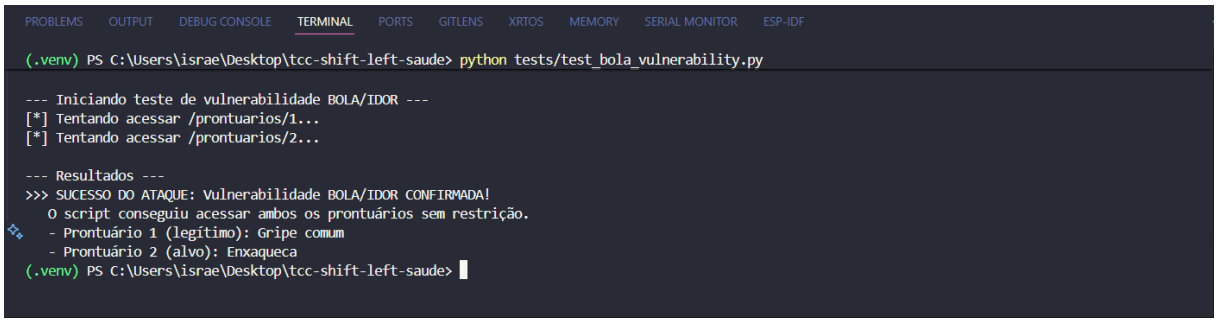
Uma falha de autorização como a BOLA/IDOR estende indevidamente esse direito a terceiros não autorizados, configurando uma clara não conformidade com a lei.

Adicionalmente, a exposição de um prontuário a quem não é o paciente ou o profissional de saúde responsável constitui uma quebra frontal do sigilo médico-paciente, indo contra o Art. 5º da Resolução CFM nº 2.314/2022. A norma exige que as plataformas assegurem a **confidencialidade e a privacidade** das informações, o que é diretamente comprometido pela ausência de um controle de acesso eficaz em nível de objeto.

4.5.2.2 Detecção com dast e correção

Para automatizar a detecção desta falha, foi desenvolvido um script de Análise Dinâmica (DAST) utilizando Pytest e Requests. O script foi projetado para simular o ataque, tentando acessar múltiplos prontuários e verificando se as respostas eram bem-sucedidas. A Figura 8 exibe a saída do script, confirmando a vulnerabilidade.

Figura 8 – Relatório do script DAST confirmando a existência da vulnerabilidade BOLA/IDOR.



Fonte: Elaborado pelo autor (2025).

A contramedida consistiu na implementação de um sistema de autenticação via tokens JWT e na adição de uma camada explícita de autorização. A nova lógica passou a exigir um token de acesso válido e a verificar se o `user_id` extraído do token corresponde ao `paciente_id` do prontuário solicitado.

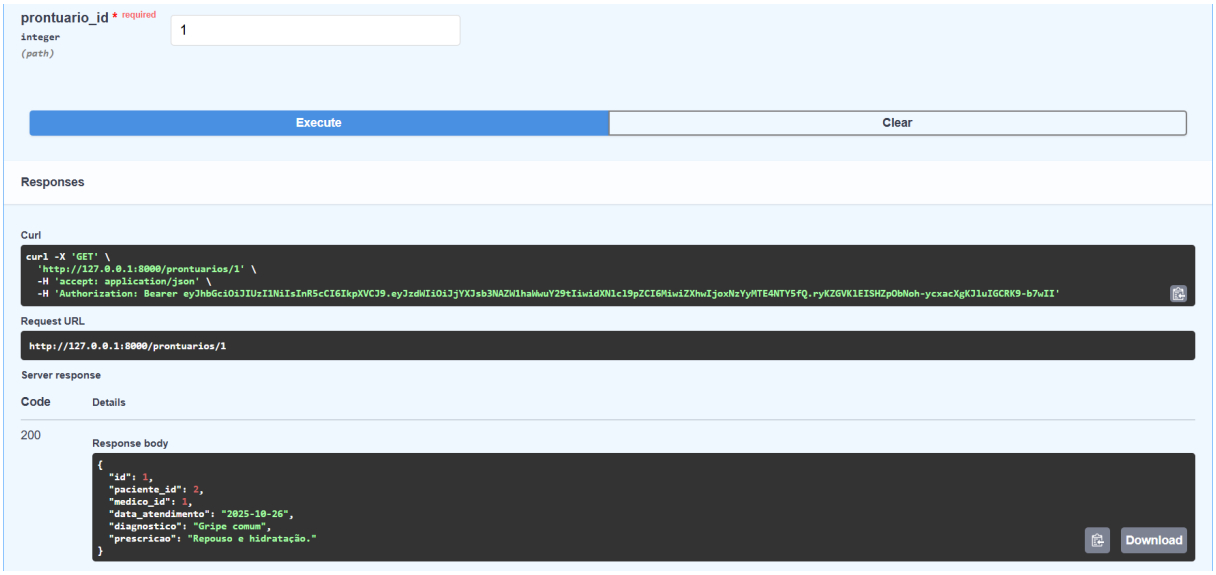
Listagem 4.4 – Código-fonte da correção da falha BOLA/IDOR

```
1 def get_prontuario_by_id(prontuario_id: int, current_user: dict =
     Depends(security.get_current_user)):
2
3     if prontuario['paciente_id'] != current_user['user_id']:
4         raise HTTPException(status_code=403, detail="Acesso
             negado...")
5
6     return Prontuario(**prontuario)
```

Fonte: Elaborado pelo autor (2025).

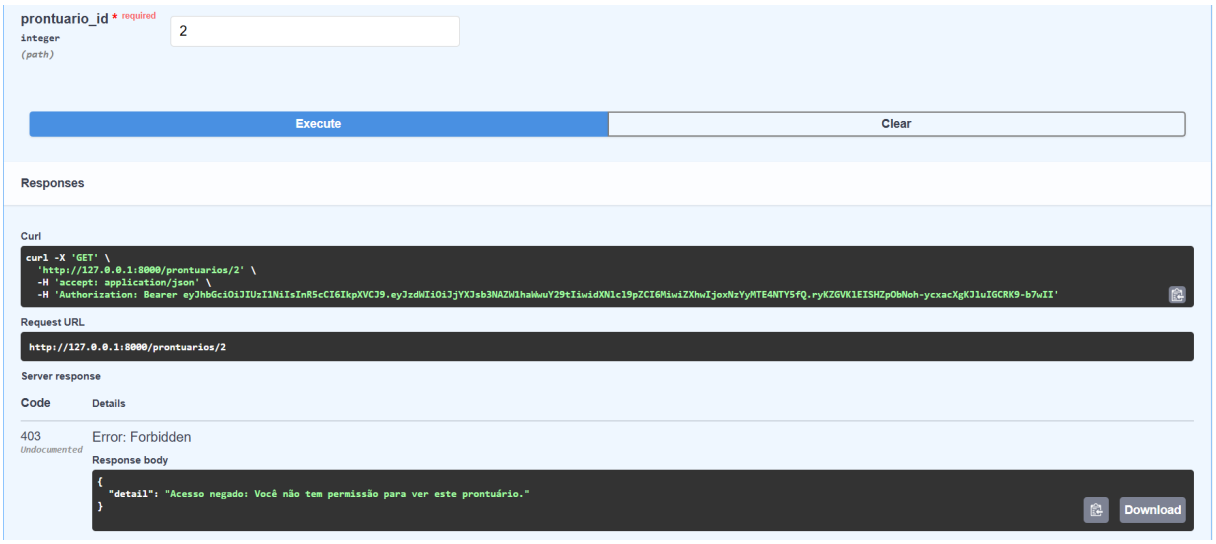
A eficácia da correção foi validada com dois testes. A Figura 9 mostra o acesso legítimo, onde um usuário autenticado consulta seu próprio prontuário (HTTP `200 OK`). Em contraste, a Figura 10 demonstra o bloqueio do ataque, com a API retornando o status `403 Forbidden` ao tentar acessar o prontuário de outro paciente.

Figura 9 – Validação da funcionalidade: acesso legítimo ao próprio prontuário.



Fonte: Elaborado pelo autor (2025).

Figura 10 – Validação da segurança: tentativa de ataque BOLA/IDOR bloqueada com erro 403.



Fonte: Elaborado pelo autor (2025).

4.5.3 Caso de estudo 3: autenticação quebrada (*broken authentication*)

A terceira vulnerabilidade estudada foi a de Autenticação Quebrada. A falha foi implementada ao se deixar a chave secreta (`SECRET_KEY`), utilizada para assinar os tokens JWT, exposta diretamente no código-fonte, uma prática conhecida como *hard-coding*. A Listagem 4.5 apresenta o código vulnerável.

Listagem 4.5 – Código-fonte da vulnerabilidade de Autenticação Quebrada

```
1 # A chave secreta esta visivel e fixa no codigo-fonte,
```

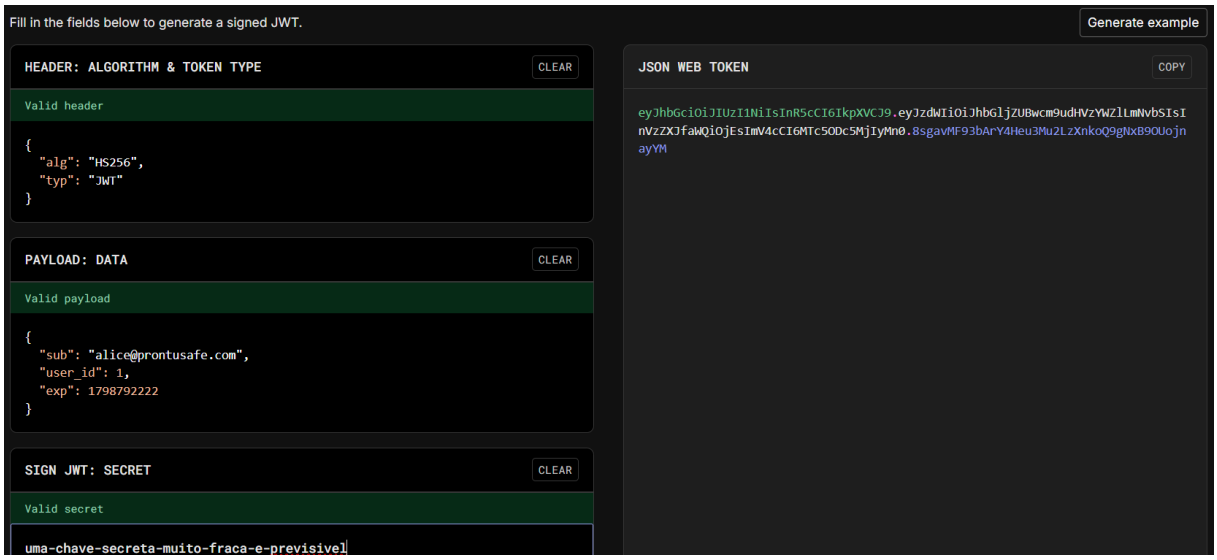
```
# representando um risco critico caso o codigo seja exposto.
SECRET_KEY = "uma-chave-secreta-muito-fraca-e-previsivel"
```

Fonte: Elaborado pelo autor (2025).

4.5.3.1 Demonstração de ataque e implicações

Com acesso à chave secreta, um atacante pode forjar tokens de acesso válidos para qualquer usuário. A Figura 11 demonstra o processo de falsificação utilizando a ferramenta online **jwt.io**. Na imagem, o `payload` de um token é alterado para representar outro usuário e, em seguida, assinado com a chave secreta exposta para gerar um token ilegítimo, mas com assinatura digital válida.

Figura 11 – Processo de falsificação de um token JWT utilizando a chave secreta exposta.



Fonte: Elaborado pelo autor (2025).

A capacidade de falsificar a identidade de um usuário por meio da exploração de uma chave secreta exposta representa um comprometimento sistêmico da segurança. Essa falha viola diretamente múltiplos princípios da LGPD, com destaque para o **Princípio da Segurança** (Art. 6º, VII), que exige a adoção de "medidas técnicas [...] aptas a proteger os dados pessoais de acessos não autorizados"(BRASIL, 2018). A prática de *hard-coding*, ou seja, manter segredos diretamente no código-fonte, não constitui uma "medida técnica apta", sendo uma falha grave de implementação.

Ademais, tal negligência compromete o **Princípio da Responsabilização e Prestação de Contas** (Art. 6º, X), que determina ao agente de tratamento o dever de:

> [...] demonstração da adoção de medidas eficazes e capazes de comprovar a observância e o cumprimento das normas de proteção de dados pessoais e, inclusive, da eficácia dessas medidas; (BRASIL, 2018, Art. 6º, X).

A exposição de uma chave crítica é uma evidência contrária à demonstração de eficácia das medidas de segurança. Em um sistema de saúde, as implicações são críticas: um atacante poderia se passar por um profissional de saúde para acessar, alterar ou exfiltrar prontuários, configurando uma violação grave do sigilo profissional preconizado pela Resolução CFM nº 2.314/2022.

4.5.3.2 Detecção com sast/dast e correção

A detecção da causa raiz (a chave exposta) foi confirmada pela ferramenta SAST Bandit, conforme o relatório da Figura 12, que identifica a presença de uma senha *hard-coded*. Para validar o impacto da vulnerabilidade, um script DAST foi executado, utilizando o token forjado. A Figura 13 mostra que a API aceitou o token, respondendo com um erro de autorização (`403`) em vez de um erro de autenticação (`401`), confirmando a falha.

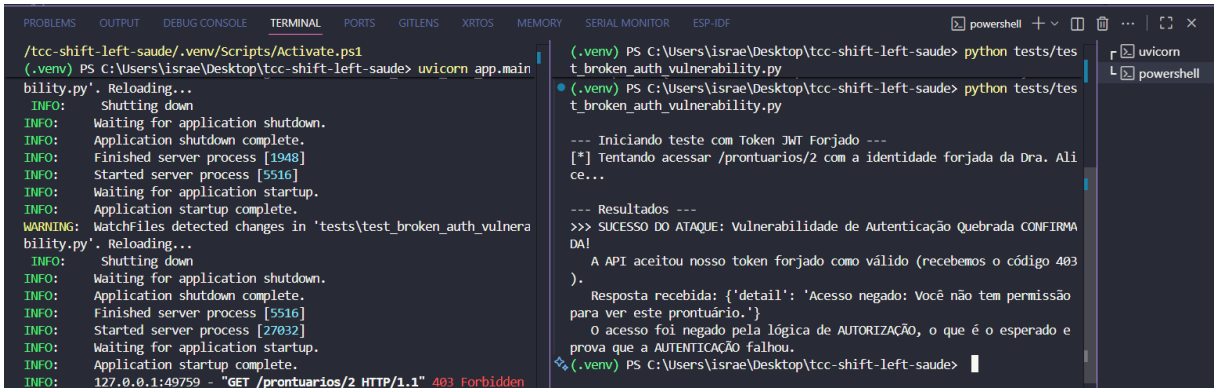
Figura 12 – Relatório do Bandit identificando a chave secreta *hard-coded*.

```
(.venv) PS C:\Users\israe\Desktop\tcc-shift-left-saude>  bandit -r app/
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.11.9
Run started:2025-11-02 22:59:24.328003

Test results:
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password:
'uma-chave-secreta-muito-fraca-e-previsivel'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b105_hardco
ded_password_string.html
   Location: app/security.py:10:13
9
10      SECRET_KEY = "uma-chave-secreta-muito-fraca-e-previsivel"
11      ALGORITHM = "HS256"
```

Fonte: Elaborado pelo autor (2025).

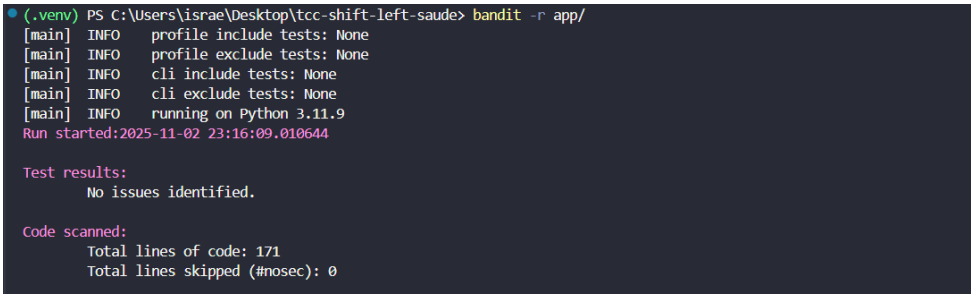
Figura 13 – Script DAST confirmando que a API aceitou o token forjado.



Fonte: Elaborado pelo autor (2025).

A contramedida consistiu em remover a `SECRET_KEY` do código-fonte e gerenciá-la como uma variável de ambiente através de um arquivo `.env`, que é ignorado pelo sistema de controle de versão. Após a correção, a Figura 14 mostra o resultado limpo da nova varredura do Bandit.

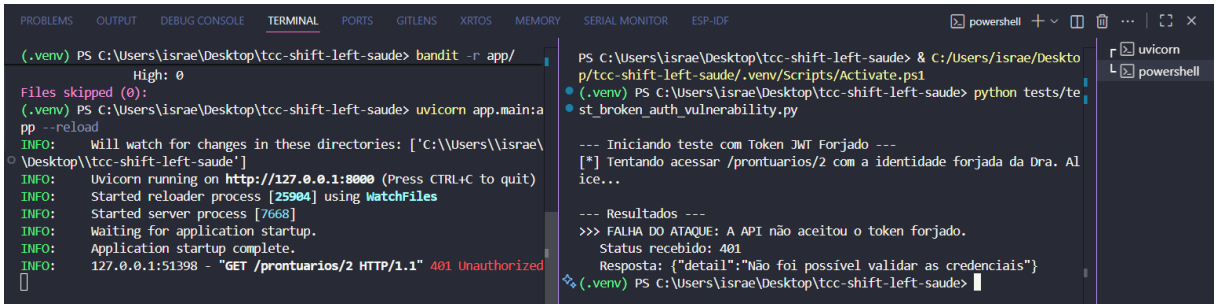
Figura 14 – Resultado do Bandit após a correção, não mais detectando a falha.



Fonte: Elaborado pelo autor (2025).

Finalmente, a Figura 15 demonstra a validação final com o script DAST. A API, agora utilizando a nova chave secreta, rejeita o token antigo e forjado, retornando o erro `401 Unauthorized` e provando que o sistema está protegido contra a falsificação.

Figura 15 – Script DAST demonstrando o bloqueio do ataque após a correção.



Fonte: Elaborado pelo autor (2025).

4.6 CRONOGRAMA DE EXECUÇÃO DO PROJETO

A execução deste trabalho ocorreu ao longo de um período aproximado de dez semanas, compreendido entre os meses de outubro e dezembro de 2025. Com o objetivo de organizar e sistematizar as atividades desenvolvidas, foi elaborado um cronograma de execução, apresentado na Tabela 5, estruturado no formato de um Gráfico de Gantt simplificado.

O cronograma distribui as principais etapas do projeto ao longo das semanas de cada mês, contemplando desde o levantamento teórico inicial até a revisão final e a preparação para a apresentação do trabalho. Essa organização permitiu o acompanhamento progressivo das atividades, assegurando coerência metodológica e alinhamento entre o planejamento e a execução do projeto.

Tabela 5 – Cronograma de execução das atividades do projeto

Atividade	**Outubro**				**Novembro**				**Dezembro**	
	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2
Levantamento e fundamentação teórica	X	X								
Planejamento do projeto e modelagem de ameaças			X							
Configuração do ambiente de desenvolvimento e implementação da API base				X						
Análise e exploração da vulnerabilidade de Injeção de SQL				X	X					
Análise e exploração da vulnerabilidade BOLA/IDOR					X					
Análise e exploração da vulnerabilidade de Autenticação Quebrada						X				
Implementação do pipeline de segurança (SAST e DAST)						X	X			
Implementação do mecanismo de notificações automatizadas							X			
Redação e consolidação da monografia								X	X	X
Revisão final e preparação para defesa										X

Fonte: Elaborado pelo autor (2025).

5 RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados obtidos com a implementação do pipeline de segurança automatizado e discute como esses resultados validam a eficácia da abordagem *Shift-Left Security* no contexto do projeto. Enquanto o capítulo anterior detalhou o ciclo de mitigação para cada vulnerabilidade, esta seção foca nos resultados da automação desse processo.

5.1 IMPLEMENTAÇÃO DO PIPELINE DE SEGURANÇA AUTOMATIZADO COM GITHUB ACTIONS

O principal resultado prático deste trabalho foi a construção de um pipeline de Integração Contínua (CI) com capacidades de segurança integradas (DevSecOps), utilizando a ferramenta GitHub Actions. O objetivo deste pipeline foi automatizar a execução das análises estática (SAST) e dinâmica (DAST) a cada alteração submetida ao repositório de código, funcionando como um "portão de qualidade"automatizado.

A configuração do pipeline, definida no arquivo <.github/workflows/security_pipeline.yml>, estabeleceu um fluxo de trabalho que executa as seguintes etapas principais: (1) configuração do ambiente Python; (2) execução da ferramenta SAST Bandit para análise do código-fonte; (3) em caso de falha no SAST, envio de uma notificação por e-mail ao desenvolvedor; e (4) caso o SAST seja bem-sucedido, a inicialização da API em um ambiente de teste para a execução dos testes de segurança dinâmicos (DAST) com Pytest.

5.2 VALIDAÇÃO DO PIPELINE EM AÇÃO

Para validar a funcionalidade do pipeline, foram simulados dois cenários: um com a submissão de código seguro e outro com a introdução intencional de uma vulnerabilidade.

5.2.1 Cenário 1: pipeline com código seguro

Com o código-fonte livre das vulnerabilidades estudadas e devidamente corrigido, o pipeline foi executado com sucesso. Todas as etapas, incluindo o scan do Bandit e os testes de segurança do Pytest, foram concluídas sem erros, resultando em uma execução bem-sucedida, marcada com o status "verde"no GitHub, conforme ilustra a Figura 16. Isso demonstra que o pipeline valida corretamente o código que está em conformidade com a política de segurança estabelecida.

Figura 16 – Execução bem-sucedida do pipeline com o código-fonte seguro.

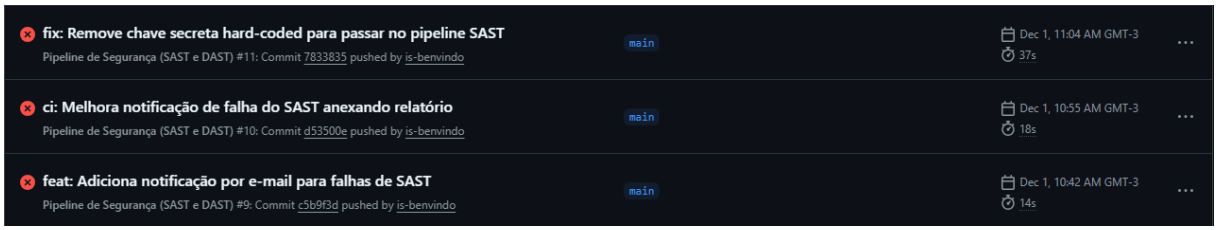
Fonte: Elaborado pelo autor (2025).

5.2.2 Cenário 2: detecção de vulnerabilidade e notificação automatizada

Para testar a capacidade de detecção do pipeline, a vulnerabilidade de Autenticação Quebrada (chave secreta *hard-coded*) foi reintroduzida no código-fonte. Ao submeter essa alteração, o pipeline foi acionado e, como esperado, falhou na etapa de SAST.

A Figura 17 demonstra o resultado, onde o GitHub Actions marca a execução como falha ("vermelha"), impedindo que o código inseguro prossiga no ciclo de desenvolvimento.

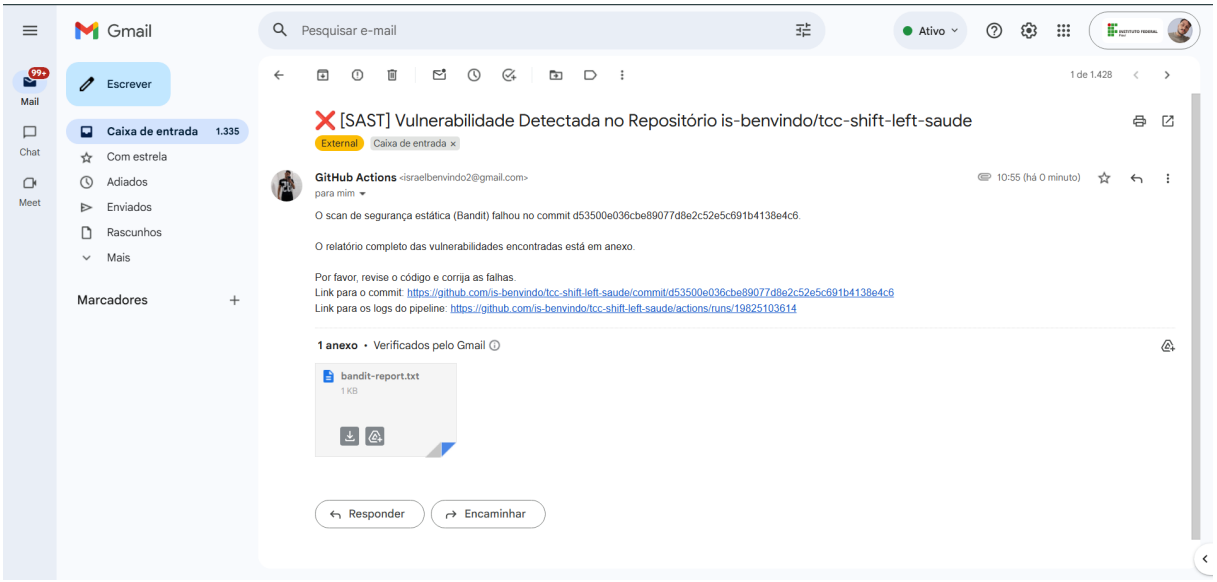
Figura 17 – Falha do pipeline após a detecção de uma vulnerabilidade pelo scan SAST.



Fonte: Elaborado pelo autor (2025).

Imediatamente após a falha, o pipeline executou a etapa de notificação. A Figura 18 exibe o e-mail gerado automaticamente e enviado ao desenvolvedor, contendo o relatório da falha em anexo. Este mecanismo garante um fluxo de comunicação contínuo e contribui para a rápida correção da vulnerabilidade, fechando o ciclo de feedback do Shift-Left.

Figura 18 – Exemplo de e-mail de notificação de falha em workflow de segurança.



Fonte: Elaborado pelo autor (2025).

5.3 DISCUSSÃO DOS RESULTADOS

Os resultados práticos obtidos confirmam a eficácia da abordagem *Shift-Left Security*. A automação do SAST e DAST em um pipeline de CI provou ser uma estratégia robusta para:

- **Reduzir a incidência de vulnerabilidades comuns**, ao barrar código inseguro antes que ele seja integrado.
- **Melhorar a qualidade e a robustez do código**, ao impor uma política de segurança de forma consistente.
- **Identificar falhas críticas de forma ágil**, reduzindo a "janela de exposição"da vulnerabilidade.
- **Gerar feedbacks contínuos e acionáveis** (através das falhas no pipeline e das notificações), permitindo que os desenvolvedores corrijam problemas rapidamente.

5.3.1 A importância da política de segurança na configuração de ferramentas

Um resultado particularmente relevante observado durante a implementação do pipeline foi a dissonância entre a classificação de risco padrão da ferramenta SAST e a política de segurança exigida pelo projeto. Conforme demonstrado, a ferramenta Bandit, em sua configuração padrão, classificava a vulnerabilidade de chave secreta exposta no código-fonte (*hard-coded secret*, CWE-259) como de severidade "Baixa".

Para o contexto deste trabalho, que simula um sistema de saúde digital e lida com dados sensíveis, tal vulnerabilidade representa um risco "Crítico".

Inicialmente, essa discrepância permitiu que o pipeline de segurança fosse concluído com sucesso mesmo na presença de uma falha grave, criando uma falsa sensação de segurança. A Figura 19 ilustra este cenário, onde o commit contendo a chave secreta não foi barrado.

Figura 19 – Execução bem-sucedida do pipeline apesar da presença de uma vulnerabilidade crítica, devido à configuração padrão da ferramenta SAST.

Fonte: Elaborado pelo autor (2025).

Isso demonstrou a necessidade de ajustar a configuração do pipeline para que a execução do Bandit falhasse ao encontrar *qualquer* issue, independentemente da severidade atribuída pela ferramenta. Essa customização alinhou o comportamento da ferramenta à política de segurança interna do projeto, que é mais rigorosa do que o padrão.

A lição fundamental deste resultado é que a segurança eficaz em um modelo DevSecOps não reside apenas na adoção de ferramentas automatizadas, mas na sua **configuração consciente e no alinhamento com os objetivos de segurança do negócio**. As organizações devem definir suas próprias políticas de risco e garantir que suas ferramentas automatizadas as reforcem, em vez de confiar cegamente nas configurações padrão.

Por fim, a conexão dos aspectos técnicos com o contexto regulatório da LGPD (BRASIL, 2018) e da Resolução CFM nº 2.314/2022 (MEDICINA, 2022) reforça que a implementação dessas práticas não é apenas uma boa prática de engenharia de software, mas um requisito para a conformidade legal no setor de saúde digital.

6 CONCLUSÃO

O presente trabalho atingiu seu objetivo ao desenvolver uma aplicação de laboratório que simulou um módulo de prontuários eletrônicos, possibilitando a avaliação prática, controlada e baseada em evidências da eficácia das estratégias de segurança do tipo *Shift-Left*. A abordagem adotada demonstrou o valor da antecipação das práticas de segurança desde as fases iniciais do ciclo de desenvolvimento de software, evidenciando sua aderência às boas práticas de engenharia de software seguro e às diretrizes de proteção de dados no contexto da saúde digital.

A utilização da linguagem Python e do *framework* FastAPI viabilizou a construção de uma API representativa de um ambiente de telemedicina. A inserção controlada de vulnerabilidades críticas permitiu a criação de um cenário realista para a aplicação das ferramentas de segurança. Nesse contexto, a análise estática (SAST), realizada por meio da ferramenta Bandit, e a análise dinâmica (DAST), por meio de scripts desenvolvidos com Pytest, mostraram-se eficazes na detecção das falhas propostas e na validação das contramedidas implementadas.

Como principal resultado deste estudo, destaca-se a implementação de um pipeline de DevSecOps por meio do GitHub Actions, responsável por automatizar o ciclo de verificação, detecção e notificação de falhas de segurança. Foi demonstrado que o pipeline é capaz de impedir a propagação de código inseguro a partir da análise estática, bem como de validar continuamente os mecanismos de segurança por meio da execução automatizada dos testes dinâmicos. A configuração de notificações automáticas contribuiu para o fechamento do ciclo de *feedback*, caracterizando um fluxo de trabalho de segurança contínuo, proativo e alinhado ao paradigma do *Shift-Left Security*.

Os resultados obtidos evidenciaram, de forma consistente, que a adoção de práticas de *Shift-Left Security* contribui não apenas para a detecção precoce de vulnerabilidades, mas também para o fortalecimento da conformidade com os requisitos estabelecidos pela Lei Geral de Proteção de Dados (LGPD) e pela Resolução CFM nº 2.314/2022. Nesse sentido, conclui-se que a integração da segurança como parte intrínseca e automatizada do ciclo de desenvolvimento representa uma estratégia fundamental para a construção de um ecossistema de saúde digital mais ético, seguro e confiável.

Como limitações deste trabalho, destaca-se o fato de a aplicação ter sido desenvolvida em ambiente de laboratório, não sendo implementada em um cenário real. Ademais, o escopo das vulnerabilidades analisadas foi restrito a um subconjunto das principais falhas descritas no OWASP API Security Top 10, o que não esgota todas as possibilidades de avaliação de segurança em sistemas de saúde digital.

7 TRABALHOS FUTUROS

O presente trabalho estabeleceu uma base metodológica sólida para a aplicação e avaliação de práticas de *Shift-Left Security* em um ambiente de laboratório controlado. A partir dos resultados obtidos, emerge um conjunto claro de oportunidades para a continuação e o aprofundamento desta pesquisa. A principal proposta para o futuro é a aplicação desta metodologia em um cenário real, com o objetivo de coletar dados empíricos para a **publicação de um artigo científico**.

As principais sugestões para trabalhos futuros, que serviriam como base para este artigo, são detalhadas a seguir:

- **Aplicação em Cenário Real e Coleta de Métricas:** A proposta central seria a implementação deste pipeline de DevSecOps em um projeto-piloto dentro de um ambiente de desenvolvimento real, como em uma clínica ou empresa de software para a saúde. Essa experiência permitiria a coleta de métricas quantitativas e qualitativas essenciais para um artigo científico, tais como:
  - O número de vulnerabilidades de criticidade alta/média detectadas pelo pipeline antes de chegarem a um ambiente de homologação.
  - O tempo médio para correção (*Mean Time to Remediate* - MTTR) de uma falha de segurança após a notificação automática.
  - Uma análise comparativa do custo de correção de uma falha encontrada na fase de codificação versus uma encontrada em fases posteriores.
  - A percepção da equipe de desenvolvimento sobre a integração das ferramentas de segurança em seu fluxo de trabalho.

  A análise desses dados empíricos forneceria uma validação robusta e publicável sobre a eficácia e o retorno sobre o investimento (ROI) da abordagem *Shift-Left*.
- **Expansão do Escopo Funcional e de Vulnerabilidades:** Ampliação da API para incluir o ciclo completo de operações de CRUD (*Create, Read, Update, Delete*) para os prontuários e a implementação de um Controle de Acesso Baseado em Papéis (RBAC) para diferenciar "Médicos"de "Pacientes". Isso permitiria o estudo de outras classes de vulnerabilidades do OWASP API Security Top 10, como *Mass Assignment* e *Security Misconfiguration*.
- **Aprimoramento dos Mecanismos de Proteção em Tempo de Execução:** Além da prevenção no desenvolvimento, poderiam ser implementados controles de segurança adicionais para a aplicação em execução, como:

  - **Rate Limiting:** Mecanismo para limitar o número de requisições a endpoints críticos (como o de login), mitigando ataques de força bruta e de negação de serviço (DoS).
  - **Web Application Firewall (WAF):** Adoção de um WAF para filtrar requisições maliciosas antes mesmo que elas cheguem à API.

- **Adoção de Análise de Composição de Software (SCA):** Integração de uma nova etapa ao pipeline de CI/CD para realizar a Análise de Composição de Software. Ferramentas de SCA escaneiam as dependências do projeto (o arquivo `requirements.txt`) em busca de bibliotecas que possuam vulnerabilidades de segurança já conhecidas (CVEs), abordando um vetor de ataque cada vez mais comum.

- **Implementação de Logs de Auditoria:** Desenvolvimento de um sistema de *logging* detalhado que registre todas as ações sensíveis (quem acessou qual prontuário e quando), um requisito fundamental para a rastreabilidade, investigações forenses e conformidade com a LGPD.

A execução dessas propostas, culminando na análise de dados de um caso de uso real, permitiria não apenas aprofundar o estudo prático sobre segurança, mas também contribuir com a comunidade acadêmica e profissional através da publicação de um artigo científico com resultados e métricas concretas.

REFERÊNCIAS

BRASIL. Lei, *Lei nº 13.709, de 14 de agosto de 2018*: Lei geral de proteção de dados pessoais (lgpd). 2018. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709.htm>. Acesso em: 30 jun. 2025.

Brasil. *Lei nº 14.510, de 27 de dezembro de 2022*. 2022. Dispõe sobre a prestação de serviços de telessaúde. Diário Oficial da União, 28 dez. 2022. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2019-2022/2022/lei/l14510.htm>. Acesso em: 1 jul. 2025.

DENCHEVA, L. *Comparative analysis of Static application security testing (SAST) and Dynamic application security testing (DAST) by using open-source web application penetration testing tools*. Dissertação (Mestrado) — Dublin, National College of Ireland, dez. 2022. Disponível em: <https://norma.ncirl.ie/id/eprint/5956>. Acesso em: 6 jul. 2025.

DHILLON, D. Developer-driven threat modeling: Lessons learned in the trenches. *IEEE Security and Privacy*, v. 9, n. 4, p. 41–47, 2011. Disponível em: <https://doi.org/10.1109/MSP.2011.47>. Acesso em: 6 jul. 2025.

DOWD, M.; MCDONALD, J.; SCHUH, J. *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. [S.l.]: Addison-Wesley Professional, 2006.

EWOH, P.; VARTIAINEN, T. Vulnerability to cyberattacks and sociotechnical solutions for health care systems: Systematic review. *Journal of Medical Internet Research*, v. 26, p. e46904, maio 2024. Disponível em: <https://www.jmir.org/2024/1/e46904/>. Acesso em: 1 jul. 2025.

FILHO, A. D. P. C. Uso de big data em saúde no brasil: perspectivas para um futuro próximo. *Epidemiologia e Serviços de Saúde*, v. 24, n. 2, p. 387–394, 2015. Disponível em: <https://doi.org/10.5123/S1679-49742015000200015>. Acesso em: 6 jul. 2025.

FRYDMAN, M. et al. Automating risk analysis of software design models. *The Scientific World Journal*, jun. 2014. Disponível em: <https://doi.org/10.1155/2014/805856>. Acesso em: 6 jul. 2025.

HORVATH, M.; ZUMERLE, D.; GARDNER, D. *Magic Quadrant for Application Security Testing*. [S.l.], 2020. Disponível em: <https://www.gartner.com/doc/reprints?id=1-1Y9S0T3&ct=200429&st=sb>. Acesso em: 6 jul. 2025.

MALIK, G.; PRASHASTI. Shift left security. *The Eastasouth Journal of Information System and Computer Science*, v. 2, n. 03, p. 219–245, abr. 2025. Disponível em: <https://doi.org/10.58812/esjscs.v2i03>. Acesso em: 6 jul. 2025.

MEDICINA, C. F. de. Resolução, *Resolução CFM nº 2.314/2022*: Define e regulamenta a telemedicina, como forma de serviços médicos mediados por tecnologias de comunicação. 2022. Publicada no D.O.U. de 05 de maio de 2022, Seção I, p. 227. Disponível em: <https://sistemas.cfm.org.br/normas/visualizar/resolucoes/BR/2022/2314>. Acesso em: 30 jun. 2025.

MOCYDLARZ-ADAMCEWICZ, M. et al. Management of onsite and remote communication in oncology hospitals: Data protection in an era of rapid technological advances. *J Pers Med*, v. 13, n. 5, p. 761, abr. 2023. Disponível em: <https://doi.org/10.3390/jpm13050761>. Acesso em: 6 jul. 2025.

OWASP Foundation. *OWASP Top 10 - 2021*. 2021. Website oficial do OWASP. Disponível em: <https://owasp.org/Top10/>. Acesso em: 6 jul. 2025.

PARGAONKAR, S. A comprehensive research analysis of software development life cycle (sdlc) agile & waterfall model advantages, disadvantages, and application suitability in software quality engineering. *International Journal of Scientific and Research Publications*, v. 13, n. 8, p. 120–124, ago. 2023. Disponível em: <https://doi.org/10.29322/IJSRP.13.08.2023.p14015>. Acesso em: 6 jul. 2025.

RANI, V. S. et al. Shift-left testing in devops: A study of benefits, challenges, and best practices. In: *2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS)*. IEEE, 2023. Disponível em: <https://doi.org/10.1109/ICACRS58579.2023.10404436>. Acesso em: 6 jul. 2025.

RAPôSO, C. F. L. et al. Lgpd - lei geral de proteção de dados pessoais em tecnologia da informação: Revisão sistemática. *RACE - Revista de Administração do Cesmac*, v. 4, p. 58–67, ago. 2019. Disponível em: <https://doi.org/10.3131/race.v4i0.1035>. Acesso em: 5 jul. 2025.

SILVA, P. d. S.; SANTOS, E. A. d. B. Python para análise de dados em segurança cibernética utilizando regex. *Revista de Engenharia e Pesquisa Aplicada*, v. 10, n. 1, p. 33–42, 2025. Disponível em: <https://doi.org/10.25286/repa.v10i1.2508>. Acesso em: 6 jul. 2025.

SMITH, L. *Shift-Left Testing*. 2001. Artigo online no Dr. Dobb's Journal. Disponível em: <https://www.drdobbs.com/shift-left-testing/184404768>. Acesso em: 1 jul. 2025.

SOARES, M. d. S. Metodologias Ágeis extreme programming e scrum para o desenvolvimento de software. *Revista de Engenharia de Software e Informação (RESI)*, v. 3, n. 1, p. 6–13, 2004. Disponível em: <https://doi.org/10.21529/RESI.2004.0301006>. Acesso em: 5 jul. 2025.

VORUGANTI, K. K. Implementing security by design practice with devsecops shift left approach. *J. Tech. Innovations*, v. 2, n. 1, fev. 2021. Disponível em: <https://doi.org/10.93153/tesiirjb7e>. Acesso em: 6 jul. 2025.