Proposta de framework conceitual para a detecção de vulnerabilidades durante o desenvolvimento de programação

Felipe Kenji Zyahana*, Anna Beatriz Santos Custódio*,
Everton Knihs*, Fabio Silva Lopes* e Ismar Frango Silveira*
*Universidade Presbiteriana Mackenzie, São Paulo, Brasil
Email: felipezyahana@gmail.com, custodioannabia@gmail.com,
tomspsp@gmail.com, flopes@mackenzie.br, ismarfrango@gmail.com

***Abstract*—Security vulnerabilities in software represent significant threats, with financial and reputational risks. Assessing security often occurs late in the Software Development Life Cycle (SDLC), resulting in costs and delays. This study proposes a framework that outlines the development of a tool aimed at advancing security validation during the implementation phase of the SDLC, promoting secure code from the outset. Objectives include vulnerability analysis, solutions, and rules for code analysis tools. The study fills a gap, providing guidance to enhance software development security.**

***Keywords*—vulnerability; framework; development; SDLC.**

***Resumo*—Vulnerabilidades de segurança em software representam ameaças sérias, com riscos financeiros e de reputação. Avaliar a segurança geralmente ocorre tardiamente no Ciclo de Vida de Desenvolvimento de Software (SDLC), acarretando custos e atrasos. Este estudo propõe um framework que estrutura a construção de uma ferramenta que visa antecipar a validação de segurança na fase de implementação do SDLC, promovendo código seguro desde o início. Objetivos incluem análise de vulnerabilidades, soluções e regras para ferramentas de análise de código. O estudo preenche uma lacuna, fornecendo orientações para aprimorar a segurança no desenvolvimento de software.**

***Palavras-chave*—vulnerabilidade; framework; desenvolvimento; SDLC .**

I. Introdução

As vulnerabilidades de segurança identificadas em sistemas e aplicações representam uma ameaça significativa, muitas das quais são amplamente conhecidas e, consequentemente, têm soluções disponíveis. Elas causam prejuízos para empresas envolvidas no desenvolvimento ou utilização de software, os riscos associados a essas vulnerabilidades podem resultar em perdas financeiras, danos à reputação, vazamento de dados confidenciais e até mesmo ações judiciais.

No entanto, com frequência, a avaliação de segurança de uma solução ocorre apenas após a conclusão do desenvolvimento, o que pode aumentar os custos de implementação e atrasar o cronograma de lançamento do sistema. Atualmente, o Ciclo de Vida de Desenvolvimento de Software (SDLC) enfatiza diversas etapas essenciais para a concepção de uma aplicação de software [1], porém a inclusão dos requisitos de segurança ocorre, comumente, em fases posteriores do processo.

Neste contexto, surge a necessidade de antecipar a validação de segurança dentro do SDLC, a fim de mitigar as vulnerabilidades já conhecidas e, ao mesmo tempo, evitar interrupções no processo de desenvolvimento. Esta pesquisa se concentra em desenvolver um framework que possa auxiliar a implementação de uma ferramenta de detecção de vulnerabilidades no código durante a fase de implementação do SDLC. Ao realizar essa validação de segurança de forma mais precoce no ciclo de desenvolvimento, espera-se que os desenvolvedores se acostumem a criar código seguro, reduzindo assim a exploração de falhas de segurança que ainda são utilizadas em ataques reais e incorporando a segurança como um componente essencial do ciclo.

Os objetivos específicos deste trabalho incluem: Analisar vulnerabilidades conhecidas, especificamente aquelas encontradas no código, que podem ser identificadas durante a implementação; Propor soluções para a implementação de código seguindo as melhores práticas de segurança; Criar regras e normas para que seja possível implementar uma ferramenta que possa realizar a análise de código durante a programação.

Este estudo tem a motivação que se fundamenta na necessidade de auxiliar a implementação de futuras ferramentas que aprimorarão a segurança no ciclo de desenvolvimento de software. Nesse contexto, a iniciativa de desenvolver o framework em questão assume uma importância incontestável, visando a promoção de um ambiente virtual mais robusto e na asseguração da efetiva integração dos princípios de segurança ao longo de todo o ciclo de vida do desenvolvimento de

software.

Observamos a existência de inúmeras vulnerabilidades previamente catalogadas, e há uma imperiosa necessidade de promover o desenvolvimento de ferramentas de apoio aos programadores, exemplificadas por nomes como Copilot, CodeWhisperer e GPT-4. No entanto, uma lacuna notável emerge nesse contexto: a ausência de um framework dedicado à concepção dessas aplicações. É nesse âmbito que este documento se insere, propondo-se a preencher essa importante lacuna, abordando a criação de um framework para tal finalidade.

II. Referêncial Teórico

O presente trabalho apresenta a sua fundamentação teórica, dividindo-a em duas seções principais: conceitos iniciais, vulnerabilidades no ambiente web e seus subtópicos, que são correlacionadas pela sequência de conteúdo apresentado. Primeiramente, serão apresentados os conceitos iniciais que compõem o cerne deste trabalho, contextualizando as principais concepções, como vulnerabilidade e suas divisões, o ciclo de vida de desenvolvimento de sistemas e a definição do conceito de análise de código.

De acordo com a pesquisa conduzida por Rodrigues et al. (2020) [2], práticas de segurança no desenvolvimento de software seguro, como teste de penetração e análise de código estática e dinâmica, não são frequentemente utilizadas por pequenos times de desenvolvimento nas organizações. Esse estudo apontou que adotar de práticas proativas durante todo o ciclo de desenvolvimento é crucial para mitigar falhas de segurança e assegurar a qualidade do software. A análise de práticas seguras demonstrou que, apesar da conscientização sobre a importância dessas práticas, há uma carência significativa na aplicação prática entre pequenos times, refletindo na necessidade de incorporar essas práticas desde as fases iniciais do desenvolvimento de software.

Além disso, a pesquisa de Prado [3] sugere que a inclusão de técnicas e ferramentas de segurança durante as fases de desenvolvimento é essencial para a prevenção de vulnerabilidades em ambientes de desenvolvimento ágeis. A implementação dessas práticas, que incluem revisões manuais e automatizadas de código, pode reduzir significativamente a exposição a riscos de segurança. O estudo enfatiza a importância de uma abordagem integrada e contínua de segurança ao longo do ciclo de vida do desenvolvimento de software, especialmente em equipes que operam sob metodologias ágeis, que frequentemente enfrentam pressões de entrega rápidas e constantes.

Por fim, a tese de Gatto [4] reforça a importância de uma arquitetura integrada de normas e frameworks de segurança para o desenvolvimento de software seguro. A pesquisa destaca que a implementação dessa arquitetura não só melhora a identificação de falhas na especificação de requisitos, como também eleva a maturidade do ciclo de vida de desenvolvimento de software seguro (S-SDLC). A aplicação prática da arquitetura demonstrou eficácia na mitigação de vulnerabilidades e na promoção de um ambiente de desenvolvimento mais seguro e confiável, evidenciando a necessidade de uma integração robusta de práticas de segurança desde o início do processo de desenvolvimento de software.

Tratando-se de segurança da informação temos que a vulnerabilidade pode ser definida como uma falha em um sistema que permite a um atacante usá-lo de uma maneira não prevista pelo projetista [5], isto é, uma vulnerabilidade se refere à situação na qual existe a possibilidade de explorar um sistema de maneira inadequada ou indevida. De maneira simples, as vulnerabilidades podem surgir a partir:

- Erros de Implementação;
- Falhas de design;
- Configurações incorretas;
- Falhas de atualização e patches;
- Engenharia Social e Manipulação.

Em geral, existem vulnerabilidades que são provenientes do desenvolvedor que não possui treinamentos em práticas de segurança e pode inadvertidamente introduzir vulnerabilidades.

O processo de desenvolvimento de software é considerado como fator principal para a existência de vulnerabilidades [6], devido à falta de detecção, as vulnerabilidades de segurança podem continuar no código durante muito tempo até serem descobertas. Tipicamente, quanto mais tarde foi feita essa detecção, mais cara fica a sua correção [7]. Logo, tornou-se imprescindível o aparecimento de mecanismos para rever eficazmente o código fonte.

Ferramentas de análise de código estático examinam o código-fonte de um programa sem executá-lo. O objetivo dessas ferramentas é extrair informações do código-fonte ou fazer julgamentos sobre ele. O uso mais comum da análise estática é em compiladores de otimização. Na verdade, a maioria das otimizações de alto nível realizadas por um compilador moderno depende dos resultados de análises estáticas, como análise de fluxo de controle e análise de fluxo de dados. Fora do contexto de compiladores, técnicas de análise estática são usadas principalmente nas áreas de métricas de software, garantia de qualidade, compreensão de programas e refatoração [6].

A. SDLC X DevSecOps

SDLC é a sigla para Software Development Life Cycle ou Ciclo de Vida de Desenvolvimento de Software, em tradução livre. É um termo que descreve o processo ou conjunto de fases pelas quais um projeto de desenvolvimento de software passa, desde a concepção até a entrega do software aos usuários

finais. O objetivo do SDLC é fornecer uma estrutura e diretrizes para o desenvolvimento de software de maneira organizada e controlada.

Os processos do SDLC podem variar de acordo com as necessidades dos desenvolvedores, produtos e/ou empresas. Algumas fases comuns podem ser destacadas [1]:

1) Planejamento
   A fase de planejamento normalmente inclui tarefas como a análise do custo-benefício, a programação, a estimativa e a alocação de recursos. A equipe de desenvolvimento coleta requisitos de várias partes envolvidas, como clientes, especialistas internos e externos e gerentes, a fim de criar um documento de especificação de requisitos do software.
2) Projeto
   Na fase de projeto, os engenheiros de software analisam os requisitos e identificam as melhores soluções para criar o software. Por exemplo, eles poderão considerar a integração de módulos pré-existentes, fazer escolhas tecnológicas e identificar ferramentas de desenvolvimento. Eles analisarão qual a melhor forma de integrar o novo software às infraestruturas de TI existentes que a organização já tiver.
3) Implementação
   Na fase de implementação, a equipe de desenvolvimento codifica o produto. Eles analisam os requisitos para identificar as tarefas de codificação menores que podem realizar diariamente para alcançar o resultado final.
4) Teste
   A equipe de desenvolvimento combina testes manuais e automatizados para identificar bugs no software. A análise de qualidade inclui testar o software para identificar erros e verificar se ele atende aos requisitos do cliente. Como muitas equipes testam o código que desenvolvem, a fase de teste pode ocorrer em paralelo à fase de desenvolvimento.
5) Implantação
   Quando as equipes desenvolvem software, eles criam e testam o código em uma cópia do software diferente daquela à qual os usuários têm acesso. O software usado pelo cliente é considerado em produção, enquanto outras cópias são consideradas como o ambiente de compilação ou ambiente de teste.
6) Manutenção
   Na fase de manutenção, entre outras tarefas, a equipe corrige bugs, soluciona problemas do cliente e gerencia as alterações do software. Além disso, a equipe monitora a performance geral do sistema, a segurança e a experiência do usuário para identificar novas formas de melhorar o software existente.

Tradicionalmente, os testes de segurança eram conduzidos como um procedimento à parte ao SDLC, o que aumenta os riscos de segurança associados àquela aplicação. Atualmente, devido à alta demanda de softwares mais seguros, a segurança foi incorporada utilizando as práticas do DevSecOps, abreviação de desenvolvimento, segurança e operações.

DevSecOps refere-se à integração de testes de segurança em todas as fases do ciclo de desenvolvimento de software. Esse conceito incorpora ferramentas e métodos que promovem a colaboração entre desenvolvedores, profissionais de segurança e equipes operacionais, visando criar software resistente às ameaças contemporâneas. Além disso, o DevSecOps assegura que práticas de segurança, como revisão de código, análise de arquitetura e testes de penetração, sejam integralmente incorporadas às iniciativas de desenvolvimento [8].

No que se refere ao tema do presente trabalho, o framework centra-se na fase de implementação, como definido na Figura 1.

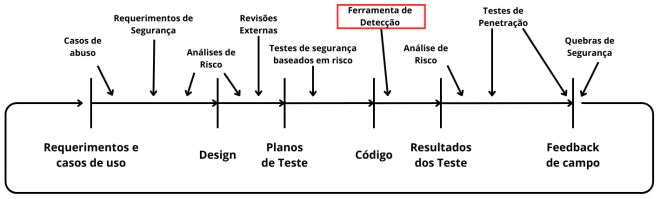

Fig. 1. O Ciclo de Vida de Desenvolvimento de Software

Para esse estudo foi considerado somente as partes específicas do SDLC, nesse caso, o processo de implementação.

B. Vulnerabilidades no ambiente web (OWASP Top Ten)

O Open Web Application Security Project (OWASP) é uma entidade sem fins lucrativos, de reconhecimento internacional, que contribui para a melhoria da segurança de softwares reunindo informações importantes que permitem avaliar riscos de segurança e combater formas de ataque através da internet [9].

Top Ten refere-se a falhas que podem ser – e são – exploradas ativamente por pessoas mal-intencionadas e causar prejuízos, mas que podem ser corrigidas pelos programadores.

De acordo com o OWASP, a maioria das vulnerabilidades da lista ocorre durante a etapa de desenvolvimento da aplicação, portanto, a recomendação é que, ainda na fase de desenvolvimento, as medidas de segurança sejam observadas pelos programadores das aplicações web como um padrão mínimo de maneira que esses sites não sejam colocados “em produção”, ou seja, à disposição dos visitantes com tais vulnerabilidades [10].

Na elaboração da lista foram levadas em consideração as experiências desenvolvidas com sucesso pelo SANS Institute, uma organização de pesquisa e educação em Segurança da Informação criada em 1989, e pelo FBI (Federal Bureau of Investigation), que publicaram uma lista de vulnerabilidades voltada a redes de computadores, além do trabalho do Web Application Security XML Project, elaborado pelo OASIS (Organization for the Advancement of Structured Information Standards), uma organização sem fins lucrativos voltada ao desenvolvimento de padrões abertos para tecnologias de informação e comunicação, que serviu para definição das categorias de vulnerabilidades a serem descritas na OWASP Top Ten. [10]

Através da Figura 2 é possível destacar o conjunto das dez vulnerabilidades mais exploradas em aplicações web de acordo com os estudos recentes do OWASP.

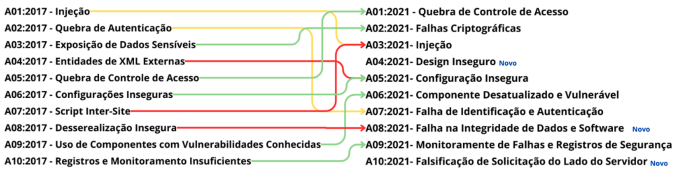

Fig. 2. Comparativo da lista Top Ten dos anos de 2017 e 2021

Basso [11] observa que, através da OWASP, "é possível encontrar informações sobre as vulnerabilidades; explicações sobre como funcionam os ataques a elas". Já Carvalho [10] expõe que o documento Top Ten não lista necessariamente as falhas mais críticas de um ambiente web, e sim as mais frequentes. Por esta razão, ao longo do tempo, a lista vai sofrendo alterações: algumas vulnerabilidades são removidas e outras colocadas em seu lugar.

É possível destacar o conjunto das dez vulnerabilidades mais exploradas em aplicações web de acordo com os estudos recentes do OWASP.

As vulnerabilidades que ocorrem no ambiente web e as principais falhas catalogadas pelo projeto OWASP (Open Web Application Security Project), (2021) indicam que vulnerabilidades poderão ser identificadas na construção do código de programação.

C. Definição de Framework

Framework é um conjunto estruturado de conceitos, práticas e ferramentas que fornece uma base para o desenvolvimento e a implementação de software. Ele oferece uma estrutura reutilizável que pode incluir bibliotecas, diretrizes, convenções

Casos	Risco
A01:2021 - Quebra de Controle de Acesso	O controle de acesso aplica políticas que garantem que os usuários não possam agir fora de suas permissões pretendidas. Falhas geralmente resultam em divulgação não autorizada de informações, modificação ou destruição de dados ou execução de funções comerciais fora dos limites do usuário.
A02:2021 - Falhas Criptográficas	A falta de criptografia representa riscos significativos para a segurança de dados em trânsito e em repouso. Por exemplo, senhas, números de cartão de crédito, registros de saúde, informações pessoais e segredos comerciais requerem proteção adicional, principalmente se esses dados estiverem sujeitos a leis de privacidade, como o Regulamento Geral de Proteção de Dados (GDPR) da União Europeia, a Lei Geral de Proteção de Dados (LGPD), ou a regulamentações, como a proteção de dados financeiros, como o Padrão de Segurança de Dados da Indústria de Cartões de Pagamento (PCI DSS).
A03:2021 - Injeção	Ocorrem quando dados não confiáveis são enviados para um interpretador como parte de um comando ou consulta. Os dados manipulados pelo atacante podem iludir o interpretador para que este execute comandos indesejados ou permita o acesso a dados não autorizados.
A04:2021 - Design Inseguro	O design inseguro é uma categoria ampla que representa diferentes vulnerabilidades, expressas como "design de controle ausente ou ineficaz". Diferenciamos entre falhas de design e defeitos de implementação por um motivo, pois têm causas e soluções diferentes. Um design seguro ainda pode ter defeitos de implementação que levam a vulnerabilidades que podem ser exploradas. Um design inseguro não pode ser corrigido por uma implementação perfeita, pois, por definição, os controles de segurança necessários nunca foram criados para se defender contra-ataques específicos. Um dos fatores que contribuem para o design inseguro é a falta de perfil de risco de negócios inerente ao software ou sistema em desenvolvimento e, portanto, a falha em determinar qual nível de design de segurança é necessário.
A05:2021 - Configuração de Segurança Incorreta	Uma boa segurança exige a definição de uma configuração segura em todos os níveis.
A06:2021 - Componente Desatualizado e Vulnerável	Componentes, tais como bibliotecas, frameworks, e outros módulos de software quase sempre são executados com privilégios elevados.
A07:2021 - Falha de Identificação e Autenticação	A confirmação da identidade do usuário, autenticação e gestão de sessões é fundamental para proteger contra-ataques relacionados à autenticação.
A08:2021 - Falha na Integridade de Dados e Software	As falhas na integridade de software e dados decorrem de código e infraestrutura que não protege contra violações de integridade. Isso ocorre, por exemplo, quando uma aplicação depende de plugins, bibliotecas ou módulos de fontes não confiáveis, repositórios e CDNs. Um pipeline de CI/CD inseguro pode permitir acesso não autorizado, código malicioso ou comprometimento do sistema. Além disso, muitas aplicações incluem atualização automática, onde as atualizações são baixadas sem verificação de integridade e aplicadas à aplicação confiável. Atacantes podem inserir suas próprias atualizações para serem distribuídas e executadas. Outro exemplo ocorre quando objetos ou dados são codificados em estruturas que um atacante pode visualizar e modificar, ficando vulneráveis à deserialização insegura.

Casos	Risco
A09:2021 - Monitoramento de Falhas e Registros de Segurança	Esta categoria visa ajudar a detectar, aumentar a gravidade e responder a violações ativas. Sem registro e monitoramento, as violações não podem ser detectadas.
A10:2021 - Falsificação de Solicitação do Lado do Servidor	Vulnerabilidades SSRF (Server-Side Request Forgery) ocorrem sempre que uma aplicação web está buscando um recurso remoto sem validar a URL fornecida pelo usuário. Isso permite que um atacante force a aplicação a enviar uma solicitação manipulada para um destino inesperado, mesmo quando protegido por um firewall, VPN ou outro tipo de lista de controle de acesso à rede (ACL). À medida que as aplicações web modernas oferecem recursos convenientes aos usuários finais, a busca de uma URL se torna um cenário comum. Como resultado, a incidência de SSRF está aumentando. Além disso, a gravidade das vulnerabilidades SSRF está aumentando devido aos serviços em nuvem e à complexidade das arquiteturas.

TABELA I
TABELA DE CASOS E RISCOS

e componentes para facilitar o processo de desenvolvimento de aplicativos. O objetivo de um framework é fornecer uma abstração que permite aos desenvolvedores criar software mais eficientemente, concentrando-se nas partes específicas do aplicativo em vez de lidar com detalhes comuns e repetitivos.

Frameworks podem ser específicos para determinadas linguagens de programação, domínios de aplicação ou paradigmas de desenvolvimento. Eles são projetados para promover a consistência, a modularidade e a reusabilidade do código, acelerando o processo de desenvolvimento e facilitando a manutenção do software ao longo do tempo.

Dessa forma, trata-se de normas, orientações e práticas sugeridas que auxiliam as organizações na aprimoração de seu controle de riscos relacionados à segurança cibernética [12].

III. METODOLOGIA

Este trabalho poderá ser classificado como aplicado e qualitativo, considerando-se como uma pesquisa metodológica devido à proposição de um artefato analítico ao final do estudo. Os recursos empregados foram a revisão da literatura e estudos bibliográficos e documentais.

O primeiro passo deste estudo foi a realização de uma revisão bibliográfica abrangente e aprofundada. Isso envolveu a pesquisa de artigos científicos, documentos técnicos e documentações de ferramentas existentes relacionados à detecção de vulnerabilidades durante o desenvolvimento de programas. O objetivo era estabelecer uma base sólida de conhecimento sobre as melhores práticas, as tecnologias existentes e as metodologias usadas para lidar com vulnerabilidades de segurança em códigos de software. A revisão bibliográfica foi fundamental para identificar lacunas no campo e para compreender as necessidades específicas que nosso framework deveria abordar.

Após a revisão bibliográfica, as informações coletadas foram cuidadosamente estruturadas e sistematizadas. Isso envolveu a organização de conceitos-chave, abordagens, metodologias e tecnologias relevantes. As informações foram categorizadas e catalogadas de maneira lógica, a fim de estabelecer um alicerce sólido para o desenvolvimento do framework. Isso também permitiu a identificação de padrões e tendências emergentes que poderiam ser incorporados no design do framework.

A etapa subsequente envolveu a construção de uma tabela de requisitos detalhados para o framework. Essa tabela foi elaborada com base nas necessidades identificadas durante a revisão bibliográfica e nas informações estruturadas anteriormente. Cada requisito foi cuidadosamente definido, priorizado e documentado. Isso proporcionou uma visão clara das funcionalidades, recursos e capacidades que o framework precisaria oferecer para abordar as vulnerabilidades de segurança no desenvolvimento de software.

Com a tabela de requisitos estabelecida, a equipe de pesquisa partiu para o desenvolvimento do framework. Isso envolveu a criação de um ambiente de desenvolvimento, a escolha das linguagens de programação e tecnologias apropriadas, e a implementação das funcionalidades necessárias.

A última etapa deste estudo envolveu a escrita e a documentação final da pesquisa. Isso incluiu a elaboração de um relatório abrangente que descreveu todo o processo, desde a revisão bibliográfica até o desenvolvimento do framework. O relatório documentou as descobertas, as soluções implementadas e os resultados obtidos. Além disso, foram fornecidas diretrizes para a utilização do framework, juntamente com exemplos e casos de uso práticos. A documentação final serviu como um guia abrangente para aqueles que desejam utilizar o framework para melhorar a segurança no desenvolvimento de software.

No geral, o desenvolvimento deste framework para a detecção de vulnerabilidades durante o desenvolvimento de programação envolveu uma abordagem metodológica abrangente que combinou pesquisa teórica sólida. O resultado foi um framework valioso para a comunidade de desenvolvimento de software, fornecendo uma abordagem eficaz para a mitigação de riscos de segurança em códigos de software.

IV. ESTRUTURAÇÃO E SISTEMATIZAÇÃO DOS REQUISITOS DE IMPLEMENTAÇÃO

A estruturação e sistematização dos requisitos considerados importantes para detecção de vulnerabilidades no desenvolvimento da programação pode ser determinada a seguir:

Primeiramente é essencial considerar a integração do plugin em ambientes de desenvolvimento como IDEs amplamente uti-

lizadas. A integração permitirá que os desenvolvedores utilizem a ferramenta de forma contínua e sem interrupções durante o processo de codificação. Esta IDE foi sugerida para este trabalho porque fornece um conjunto de APIs que permitem recursos de edição inteligentes para diferentes linguagens de programação por meio de extensões de linguagem.

O ambiente de desenvolvimento da sugestão provê um protocolo de servidor de linguagem (Language Protocol Server - LSP) [13], que padroniza a comunicação entre ferramentas de linguagem e editor de código. Dessa forma, os servidores de linguagens podem ser implementados em qualquer linguagem, para esse projeto utilizaremos a implementação em Python para a validação de códigos em Python [14].

O protocolo de servidor de linguagem possui alguns dos requisitos essenciais para a implementação do plug in, com a análise léxica e sintática. A análise léxica é o processo de dividir o código-fonte em tokens, como palavras-chave, identificadores, números e símbolos, de acordo com as regras da linguagem. Essa análise identifica erros de tokenização, como caracteres inválidos ou sequências malformadas. A análise sintática é o processo de analisar a estrutura gramatical do código-fonte para verificar se ele está bem formado de acordo com as regras da linguagem. Isso envolve a construção de uma árvore de sintaxe ou outra representação estruturada do código. Durante a análise sintática, erros de sintaxe, como falta de ponto e vírgula ou parênteses desbalanceados, são detectados. Os servidores de linguagem que suportam recursos avançados podem implementar essas análises para fornecer funcionalidades como realce de sintaxe, detecção de erros em tempo real, sugestões de correção e assistência de código inteligente [15].

Com as análises realizadas podemos usá-las para executar a análise estática é uma técnica de análise de programas que examina o código fonte ou o código objeto sem executá-lo, examinando a estrutura, regras de formação e possíveis problemas potenciais. Nesse contexto utilizaremos as análises léxicas e sintáticas para verificar a estrutura. Além das análises, é necessário ter um banco de dados com padrões de códigos vulneráveis para ser possível a comparação do código implementado pelo usuário.

Considera-se como contribuição deste estudo esta tabela de requisitos como necessária para uma futura implementação de um plugin para a validação de vulnerabilidades em tempo real. A seguir, apresenta-se a Estruturação dos Requisitos de Implementação e detecção de vulnerabilidades, identificando instrumentos para análise da construção de um código de programação:

Requisitos	**Estrutura para detecção**	**Descritivo para construção da ferramenta**
Integração com a IDE	Visual Studio Code	A integração com a IDE permite que a ferramenta possa ser usada amplamente pelos usuários
Linguagem	Python	Por ser uma linguagem amplamente utilizada, utilizaremos na implementação e na validação
Erros em tempo real	Protocolo de Servidor de Linguagem (Language Protocol Server - LSP)	A detecção de erros em tempo real permite que o usuário saiba em tempo de codificação qual o erro está cometendo
Análise Léxica, Sintática e Semântica	Protocolo de Servidor de Linguagem (Language Protocol Server - LSP)	As análises léxica, sintática e semântica realizadas pelo LSP, respectivamente, dividem o código ou texto em tokens, verificam a estrutura gramatical correta e verificam o significado e o contexto das construções. Com o resultado dessas análises, a ferramenta terá o contexto necessário para que seja realizada posteriormente a análise estática do código.
Análise Estática	Protocolo de Servidor de Linguagem (Language Protocol Server - LSP)	Utilizando as análises previamente realizadas é possível implementar a análise estática usando LSP. Essa análise permite a deteção de vulnerabilidades no código fonte.
Base de Vulnerabilidades	Arquivo JSON	Uma base de códigos vulneráveis a fim de comparar com o código fornecido pelo usuário.

TABELA II
TABELA DE REQUISITOS

A. *Framework*

O framework proposto na Figura 3 foi desenvolvido com base em padrões reconhecidos e nas melhores práticas nas organizações, visando aprimorar a detecção e mitigação de vulnerabilidades de segurança em projetos de desenvolvimento de software.

Este framework é uma ferramenta versátil que integra-se diretamente com ambientes de desenvolvimento populares, como o Visual Studio Code, e utiliza a análise estática do Protocolo de Servidor de Linguagem (Language Protocol Server - LSP) para identificar potenciais vulnerabilidades de segurança em tempo real. Além disso, o framework incorpora uma base de vulnerabilidades abrangente armazenada em um arquivo JSON, permitindo a comparação e identificação de possíveis vulnerabilidades no código-fonte do projeto em desenvolvimento.

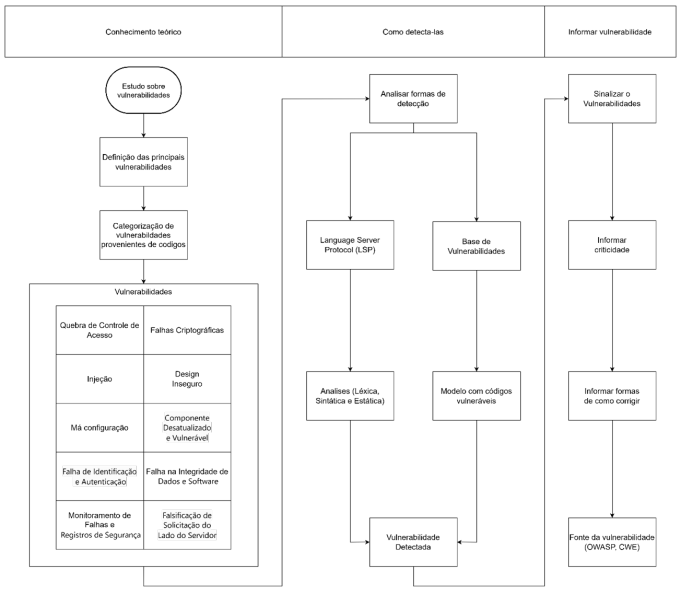

Fig. 3. Framework proposto

1) Aplicação do framework no OWASP: A segurança de software é uma preocupação fundamental no desenvolvimento de aplicativos e sistemas modernos. Vulnerabilidades de segurança podem ter consequências devastadoras, incluindo a exposição de informações confidenciais, comprometimento de dados e potencial acesso não autorizado. Nesse contexto, a detecção precoce de vulnerabilidades é crucial.

O LSP é capaz de analisar o código-fonte em busca de uma ampla gama de vulnerabilidades, incluindo aquelas relacionadas a Quebra de Controle de Acesso (A01), Falhas Criptográficas (A02), Injeção (A03), Design Inseguro (A04), Configuração Insegura (A05), Componente Desatualizado e Vulnerável (A06), Falha de Identificação e Autenticação (A07), Falha na Integridade de Dados e Software (A08), Monitoramento de Falhas e Registros de Segurança (A09) e Falsificação de Solicitação do Lado do Servidor (A10).

Este framework de detecção oferece uma abordagem holística para fortalecer a segurança do desenvolvimento de software, permitindo que desenvolvedores identifiquem e corrijam vulnerabilidades em tempo real, contribuindo assim para a construção de sistemas mais seguros e resilientes.

1) Quebra de Controle de Acesso (A01): O LSP (Language Server Protocol) analisa a falta de controle de acesso realizando uma verificação estática do código para identificar se permissões e autenticações foram aplicadas corretamente. Ele mapeia funções e padrões de autorização, como endpoints REST que exigem validação de tokens ou roles, verificando se possuem proteção adequada. Em frameworks como Django, Rails ou Spring, o LSP também identifica funções críticas que precisam de autenticação e verifica se elas estão protegidas. Além disso, o LSP detecta acessos diretos a recursos sensíveis, sinalizando alertas quando validações de segurança estão ausentes. Ele também compara as permissões do código com regras de controle de acesso padrão, como RBAC (Controle de Acesso Baseado em Funções) ou ABAC (Controle de Acesso Baseado em Atributos), e sinaliza permissões excessivamente permissivas, como acessos irrestritos (public ou *).
2) Falhas Criptográficas (A02): Para que o LSP identifique falhas criptográficas (A02), ele faz uma análise estática, verificando o uso de algoritmos criptográficos e a proteção de dados sensíveis. O LSP procura algoritmos considerados inseguros, como MD5 e SHA-1, alertando o desenvolvedor e recomendando alternativas mais seguras, como SHA-256 para hashing e AES-256 para criptografia simétrica. O LSP também identifica onde dados sensíveis, como senhas ou números de cartão, são armazenados ou transmitidos sem criptografia adequada, gerando alertas e sugerindo o uso de métodos seguros, como bcrypt para senhas. Baseado em boas práticas criptográficas, ele orienta o desenvolvedor para corrigir possíveis falhas, fortalecendo a segurança antes da fase de produção.
3) Injeção (A03): Para identificar vulnerabilidades de injeção (A03), o LSP analisa o código em busca de padrões comuns em operações de entrada de dados, especialmente onde informações do usuário são diretamente utilizadas em comandos SQL ou de sistema. O LSP procura por construções que concatenam entradas de usuário com consultas SQL ou comandos, como em strings ou variáveis sem tratamento adequado, que representam um risco elevado de injeção. Quando encontra esses padrões, o LSP verifica se a entrada está sendo sanitizada, buscando a presença de práticas seguras, como prepared statements e bind parameters em consultas SQL, ou funções de sanitização em comandos de sistema. Ele alerta o desenvolvedor caso detecte trechos sem essas

proteções e sugere o uso de prepared statements ou bibliotecas de segurança, que reduzem as chances de injeções. Para maximizar a precisão, o LSP compara o código com uma base de dados de padrões inseguros, garantindo que as entradas sejam tratadas adequadamente e aumentando a resiliência da aplicação contra ataques de injeção

4) Design Inseguro (A04): Para detectar um design inseguro (A04), o LSP realiza uma análise estática voltada para identificar áreas do código onde práticas fundamentais de segurança, como controle de acesso e restrições, não estão implementadas. Ele examina trechos que exigem proteção — como endpoints ou métodos de manipulação de dados sensíveis — e verifica se há camadas de controle de acesso ou permissões aplicadas. O LSP também busca por estruturas de código que devem aplicar lógica de perfil de risco, analisando se esses trechos contêm verificações de permissão de usuário, como autenticação e autorização. Ao detectar ausência de restrições de segurança ou controle de acesso granular, o LSP alerta o desenvolvedor e recomenda a inclusão de práticas seguras, como definição de níveis de permissão ou a implementação de controles de acesso baseados em perfis de usuário.
5) Configuração Insegura (A05): Para identificar configurações inseguras (A05), o LSP examina o código e as configurações do ambiente em busca de práticas que possam expor o sistema a riscos. Ele verifica configurações críticas, como permissões em diretórios sensíveis, definições de segurança no servidor e protocolos de comunicação, como HTTPS para proteção de dados em trânsito. O LSP monitora configurações padrão ou permissivas, alertando o desenvolvedor quando detecta definições inseguras, como permissões excessivas em arquivos ou a ausência de criptografia nas comunicações. Ao encontrar configurações que podem comprometer a segurança, o LSP sugere correções imediatas, como aplicar restrições adequadas aos diretórios ou configurar HTTPS por padrão. O plugin também consulta uma base de dados com diretrizes de segurança para garantir que os padrões recomendados estejam de acordo com as melhores práticas, ajudando a assegurar que o sistema seja protegido contra explorações relacionadas a configurações inseguras.
6) Componente Desatualizado e Vulnerável (A06): Para que o Language Server Protocol (LSP) identifique e sugira atualizações de componentes desatualizados ou vulneráveis, ele acessa uma base de dados atualizada com vulnerabilidades conhecidas, armazenada em JSON, que lista versões seguras e inseguras de bibliotecas. Fontes como a NVD (National Vulnerability Database) fornecem essas informações, permitindo que o LSP monitore os componentes usados pelo desenvolvedor.
   Durante a análise do código-fonte, o LSP verifica as dependências listadas nos arquivos de configuração do projeto, como requirements.txt no Python ou package.json no Node.js. Ao identificar uma versão desatualizada, ele alerta o desenvolvedor na IDE e sugere uma versão segura. Por exemplo, ao detectar que a biblioteca requests==2.18.0 está desatualizada e vulnerável, o LSP sugere a atualização para uma versão segura como requests==2.31.0.
7) Falha de Identificação e Autenticação (A07): A validação de Falhas de Identificação e Autenticação (A07) pelo plugin envolve a análise do código para garantir a implementação de autenticação forte e gerenciamento seguro de sessões. O plugin verifica se as rotinas de login exigem senhas complexas e se existem proteções contra ataques de força bruta, como bloqueios após tentativas falhas.
8) Falha na Integridade de Dados e Software (A08): Na aplicação do framework no contexto da Falha na Integridade de Dados e Software (A08), o Language Server Protocol (LSP) é essencial para a análise de segurança do código-fonte. O LSP verifica se o código depende de módulos ou bibliotecas de fontes confiáveis, garantindo que componentes seguros sejam utilizados. Além disso, o LSP identifica padrões de desserialização insegura, que podem introduzir vulnerabilidades. Ele analisa o fluxo de dados no código e sugere a adoção de medidas de validação e sanitização antes da desserialização, caso encontre práticas arriscadas.
9) Monitoramento de Falhas e Registros de Segurança (A09): O plugin, integrado ao protocolo de servidor de linguagem (LSP), realiza uma análise contínua do código-fonte em busca de práticas de monitoramento e registro de segurança. Durante a verificação, ele identifica a presença ou ausência de mecanismos de auditoria, essenciais para a detecção de falhas e atividades suspeitas.
10) Falsificação de Solicitação do Lado do Servidor (A10): Na aplicação real do LSP para prevenir a Falsificação de Solicitação do Lado do Servidor (SSRF), o protocolo analisa o código-fonte para identificar como as URLs fornecidas pelo usuário são tratadas nas requisições de servidor. Durante essa análise, o LSP examina o fluxo de dados para verificar se existem validações adequadas para as URLs, como a utilização de listas brancas que permi-

tam apenas domínios confiáveis. Além disso, o LSP pode sugerir práticas recomendadas para mitigação de SSRF, como a implementação de verificações que garantam que a URL não aponte para recursos internos ou serviços sensíveis que não deveriam ser acessíveis externamente. Por exemplo, se o código aceitar entradas do usuário que podem incluir URLs, o LSP pode recomendar a utilização de expressões regulares para validar se a URL segue um formato esperado e não contém padrões que poderiam levar a requisições maliciosas.

V. Considerações Finais e Trabalhos Futuros

Este estudo tem a motivação que se fundamenta na necessidade de auxiliar a implementação de futuras ferramentas que aprimorem a segurança no ciclo de desenvolvimento de software. Nesse contexto, a iniciativa de desenvolver o framework em questão assume uma importância incontestável, visando à promoção de um ambiente virtual mais robusto e à efetiva integração dos princípios de segurança ao longo de todo o ciclo de vida do desenvolvimento de software.

Apresentou-se um framework na detecção de diversas vulnerabilidades, desde problemas de controle de acesso até questões relacionadas à criptografia, injeção, design inseguro e muito mais. A integração com a IDE e a capacidade de oferecer feedback em tempo real proporcionam aos desenvolvedores uma ferramenta para melhorar a segurança de seus aplicativos desde as fases iniciais de desenvolvimento. Ademais, pode-se observar, através de [16], que a detecção de vulnerabilidades de forma preventiva proporcionou mais segurança, proteção para a aplicação, diminui custos com retrabalhos e permitiu agilidade na entrega.

O framework proposto neste trabalho oferece um meio de identificar e corrigir vulnerabilidades, visando a fortalecer a resiliência dos sistemas de software em um ambiente de ameaças em constante evolução. O uso da ferramenta idealizada no framework coopera para a criação de aplicativos mais seguros, protegendo os dados sensíveis dos usuários e garantindo a conformidade com regulamentações de privacidade e segurança.

A. Trabalhos Futuros

Para fortalecer a aplicabilidade e eficácia do framework proposto, este estudo sugere várias direções para futuras pesquisas e implementações. Essas direções incluem melhorias nas funcionalidades do framework, testes em contextos variados e integrações adicionais com ferramentas de desenvolvimento de software. A seguir, são delineadas algumas sugestões concretas para o aprimoramento do framework:

1) **Expansão das Capacidades de Detecção de Vulnerabilidades**: Inicialmente, recomenda-se estender a cobertura do framework para detectar um conjunto mais amplo de vulnerabilidades. Isso pode incluir, por exemplo, a identificação de ataques de dia zero, análise de vulnerabilidades específicas a linguagens de programação ou ambientes distintos, e melhorias na precisão da detecção de ameaças em tempo real.
2) **Testes em Ambientes Diversificados**: Realizar testes do framework em múltiplos contextos, como sistemas de diferentes setores (financeiro, saúde, comércio eletrônico), permitirá avaliar sua adaptabilidade e eficácia em cenários com perfis de segurança variados. Isso também ajudará a entender como as vulnerabilidades comuns a cada setor podem ser melhor detectadas e mitigadas.
3) **Aprimoramento de Feedback ao Desenvolvedor**: Um ponto essencial é a melhoria na interface e no sistema de feedback do framework. Estudos futuros podem investigar como fornecer feedback mais contextualizado e personalizado ao desenvolvedor, destacando as vulnerabilidades mais críticas e sugerindo correções específicas com base no código-fonte. Isso incluiria, por exemplo, a introdução de insights sobre melhores práticas de codificação segura e referências automatizadas a documentação de segurança.
4) **Avaliação de Performance e Eficiência**: Por fim, sugere-se uma análise rigorosa da performance do framework em relação ao tempo de execução e consumo de recursos. Esse estudo permitirá ajustes que reduzam o impacto do framework no desempenho da aplicação ou IDE, especialmente em projetos maiores ou mais complexos.

Essas direções para trabalhos futuros visam consolidar o framework como uma ferramenta prática e indispensável para desenvolvimento seguro de software, acompanhando as evoluções tecnológicas e a crescente necessidade de segurança digital.

Referências

[1] Amazon Web Services, “O que é o sdlc (ciclo de vida de desenvolvimento de software)?” https://aws.amazon.com/pt/what-is/sdlc/, 2023, acesso em: 29/10/2023.
[2] J. Rodrigues, K. Cordovil, O. O. Junior, and R. Torres, “Investigação das práticas proativas de desenvolvimento de software seguro adotadas por pequenos times de desenvolvimento,” in *Anais do XLVIII Seminário Integrado de Software e Hardware*. Porto Alegre, RS, Brasil: SBC, 2021, pp. 241–250. [Online]. Available: https://sol.sbc.org.br/index.php/semish/article/view/15828
[3] L. C. M. C. Santos, E. P. V. Prado, and M. L. Chaim, “Técnicas e ferramentas para detecção de vulnerabilidades em ambientes de desenvolvimento Ágil de software,” *Brazilian Journal of Development*, vol. 6, no. 6, pp. 53 577–53 594, 2020.
[4] D. de Oliveira Gatto, “Arquitetura integrada de normas e frameworks de desenvolvimento de software seguro aplicada para apoiar a identificação de falhas na especificação de requisitos,” Tese de Doutorado, Universidade Nove de Julho (UNINOVE), 2024.

[5] C. Anley, *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*, 2nd ed. Wiley Publishing, Inc., 2007.
[6] A. Sotirov, "Automatic vulnerability detection using static source code analysis," Master's thesis, University of Alabama, 2005, acesso em: 29/10/2023.
[7] B. Chess and G. McGraw, "Static analysis for security," *IEEE Security & Privacy*, vol. 2, no. 6, pp. 76–79, 2004, acesso em: 29/10/2023.
[8] Amazon Web Services, "O que é devsecops?" https://aws.amazon.com/pt/what-is/devsecops/, 2023, acesso em: 27/11/2023.
[9] OWASP, "Owasp top ten – 2021. the ten most critical web application security risks," https://owasp.org/www-project-top-ten/, 2021, acesso em: 29/10/2023.
[10] A. Carvalho, "Segurança de aplicações web e os dez anos do relatório owasp top ten: O que mudou?" https://www.fatecsaocaetano.edu.br/fascitech/index.php/fascitech/article/download/85/84, 2014, acesso em: 27/11/2023.
[11] T. Basso, "Uma abordagem para avaliação da eficácia de scanners de vulnerabilidades em aplicações web," Master's thesis, Dissertação e Apresentação de Mestrado, 2010, acesso em: 29/10/2023.
[12] IBM, "O que é o nist cybersecurity framework?" https://www.ibm.com/br-pt/topics/nist, 2023, acesso em: 29/10/2023.
[13] Microsoft Corporation, "Language extensions overview," https://code.visualstudio.com/api/language-extensions/overview, 2023, acesso em: 29/10/2023.
[14] Microsoft, "Python extension template," https://code.visualstudio.com/api/advanced-topics/python-extension-template, 2022, acesso em: 29/10/2023.
[15] ——, "Language server extension guide," https://code.visualstudio.com/api/language-extensions/language-server-extension-guide, 2023, acesso em: 29/10/2023.
[16] F. Sandrini, "Análise de vulnerabilidades de segurança computacional com a implementação do desenvolvimento," 2019, monografia de Graduação em Ciência da Computação, São Paulo, 2019.