Matheus Felipe Correa da Silva
Henrique Fialho Cardoso

Segurança na Injeção de Dependência: Melhores Práticas em Sistemas Orientados a Objetos

Belo Horizonte

2025

Matheus Felipe Correa da Silva
Henrique Fialho Cardoso

Segurança na Injeção de Dependência: Melhores Práticas em Sistemas Orientados a Objetos

Projeto de Pesquisa apresentado na disciplina Introdução à Pesquisa em Informática do curso de Engenharia de Software da Pontifícia Universidade Católica de Minas Gerais.

Belo Horizonte
2025

SUMÁRIO

RESUMO

A injeção de dependência é uma técnica fundamental no desenvolvimento de sistemas orientados a objetos, porém sua implementação inadequada pode introduzir vulnerabilidades críticas de segurança. Este projeto investiga os riscos associados às diferentes abordagens de injeção (construtor, setter e interface) em frameworks como Spring e Apache Struts, analisando casos reais de vulnerabilidades documentadas (SECURITY, 2017; SECURITY, 2022). Através de revisão sistemática da literatura e análise qualitativa de vulnerabilidades, o estudo propõe diretrizes para mitigar riscos de injeção maliciosa. Os resultados preliminares indicam que a violação do Princípio da Inversão de Dependência e o acoplamento excessivo são fatores determinantes nas vulnerabilidades analisadas, reforçando a necessidade de práticas como validação rigorosa de dependências e uso de imutabilidade.

Palavras-chave: Injeção de Dependência, Segurança em Software, Sistemas Orientados a Objetos, Vulnerabilidades CVE, Princípios SOLID.

1 INTRODUÇÃO

A injeção de dependência (ID) consolidou-se como padrão essencial no desenvolvimento de sistemas orientados a objetos modernos, promovendo baixo acoplamento e alta testabilidade. Contudo, sua implementação inadequada tem sido fonte de vulnerabilidades críticas, como demonstrado em incidentes de grande impacto como o vazamento de dados da Equifax em 2017, decorrente de falha no Apache Struts (SECURITY, 2017).

O problema central deste estudo reside na lacuna entre a adoção técnica da ID e as práticas de segurança necessárias para prevenir injeções maliciosas. Frequentemente, desenvolvedores implementam padrões de injeção sem considerar implicações de segurança, criando vetores para exploração através da substituição de dependências legítimas por objetos maliciosos.

Justifica-se esta pesquisa pelo aumento expressivo de vulnerabilidades relacionadas a frameworks de ID - somente em 2022, mais de 15 CVEs críticos foram registrados para o Spring Framework. A ausência de diretrizes consolidadas para implementação segura representa risco sistêmico para organizações.

Metodologicamente, adota-se abordagem qualitativa com três eixos: Revisão sistemática de vulnerabilidades documentadas em CVE; Análise comparativa das abordagens de injeção; E a proposição de diretrizes fundamentadas em princípios SOLID.

Este trabalho está organizado em quatro seções principais. A Seção 2 apresenta o referencial teórico sobre injeção de dependência e segurança em POO. A Seção 3 detalha a metodologia de análise de vulnerabilidades. A Seção 4 discute os resultados e diretrizes propostas, seguida pelas considerações finais.

1.1 Objetivos

O objetivo geral desse projeto é propor diretrizes seguras para a utilização de injeção de dependência em sistemas orientados a objetos, com o intuito de mitigar riscos associados a ataques por injeções maliciosas.

1.1.1 Objetivos específicos

Os objetivos específicos deste projeto são:

1. Identificar casos de vulnerabilidade decorrentes do uso indevido ou inseguro da injeção de dependência em sistemas orientados a objetos;

2. Analisar casos reais e teóricos de falhas de segurança causadas por injeções maliciosas;

3. Comparar diferentes abordagens de injeção de dependência (por construtor, por método setter e por interface) quanto aos seus impactos na segurança de sistemas orientados a objetos.

2 REVISÃO BIBLIOGRÁFICA

Esta seção apresenta uma revisão das principais bases teóricas e estudos relacionados ao tema central deste trabalho, com o objetivo de contextualizar os conceitos utilizados e evidenciar lacunas ou desafios presentes na literatura. Inicialmente, discorre-se sobre os fundamentos técnicos da injeção de dependência (ID) e suas implicações no desenvolvimento seguro de sistemas orientados a objetos. Em seguida, são abordados os aspectos conceituais e operacionais da tecnologia blockchain, com ênfase nos contratos inteligentes e nas vulnerabilidades associadas à sua implementação. Por fim, são discutidos trabalhos relacionados que investigam a segurança tanto em ambientes tradicionais quanto em plataformas descentralizadas, evidenciando a convergência entre práticas de desenvolvimento seguro e a correta gestão de dependências.

2.1 Fundamentação teórica

A injeção de dependência (ID) é um padrão de projeto amplamente utilizado para reduzir o acoplamento entre componentes de software, promovendo modularidade e manutenibilidade. No entanto, sua implementação inadequada pode introduzir vulnerabilidades e comprometer a segurança do sistema.

Um dos principais sinais de má implementação é o alto acoplamento entre classes, frequentemente associado à violação do Princípio da Inversão de Dependência (DIP). Quando módulos de alto nível dependem diretamente de implementações concretas, em vez de abstrações, a flexibilidade do sistema diminui, aumentando sua suscetibilidade a falhas. Além disso, o uso excessivo de *service locators* ou contêineres de ID como "fábricas globais" reintroduz dependências ocultas, dificultando o rastreamento das relações entre componentes e elevando os riscos de erros em tempo de execução.

A tecnologia blockchain, por sua vez, consiste em um sistema distribuído de registro imutável que garante a integridade dos dados sem a necessidade de uma autoridade central (NAKAMOTO, 2008). Desde sua introdução, tem sido aplicada em diversas áreas, sendo os contratos inteligentes (*smart contracts*) uma das inovações mais relevantes. Esses contratos são programas autoexecutáveis que armazenam e transferem ativos digitais conforme regras previamente definidas (BUTERIN, 2014).

A plataforma Ethereum, lançada em 2015, é atualmente uma das mais utilizadas para a criação de contratos inteligentes, devido à sua capacidade de execução de código Turing-completo por meio da Ethereum Virtual Machine (KUSHWAHA *et al.*, 2022). Esses contratos são predominantemente escritos na linguagem Solidity e, após compilados, são implantados na blockchain, operando de forma autônoma e imutável.

Apesar de operarem em paradigmas distintos, a injeção de dependência em sistemas orientados a objetos e a segurança em blockchain compartilham desafios fundamentais relacionados à gestão de dependências. Ambos os contextos exigem mecanismos rigorosos de validação e controle para prevenir vulnerabilidades críticas. Na programação tradicional, a injeção maliciosa ocorre quando dependências não validadas são introduzidas em tempo de execução, explorando pontos como construtores expostos ou métodos *setter* públicos. De forma análoga, contratos inteligentes enfrentam riscos semelhantes ao realizarem chamadas a contratos externos ou ao importarem bibliotecas não auditadas.

A imutabilidade, frequentemente citada como vantagem da injeção por construtor por meio de campos `final` em Java, encontra seu paralelo na natureza imutável dos contratos implantados na blockchain. No entanto, essa característica, por si só, não garante segurança se as dependências subjacentes não forem devidamente validadas. O ataque à Parity Wallet em 2017 demonstrou como uma biblioteca maliciosa comprometeu contratos que dependiam dela (HERN, 2017). De forma similar, vulnerabilidades em frameworks como Spring mostraram que uma injeção de dependência mal implementada pode abrir brechas graves em sistemas empresariais.

O Princípio da Inversão de Dependência, essencial para sistemas orientados a objetos bem arquitetados, também se aplica ao desenvolvimento de contratos inteligentes. A prática de programar para interfaces, em vez de implementações concretas, é igualmente válida ao interagir com contratos externos ou oráculos em Solidity. A ausência desse cuidado pode resultar em vulnerabilidades como chamadas reentrantes, exemplificadas no famoso ataque ao DAO, onde a ordem de execução de operações permitiu a exploração do sistema.

As lições aprendidas com a injeção de dependência em sistemas orientados a objetos, como a validação rigorosa de entradas, o uso de abstrações e o monitoramento cuidadoso das dependências, são diretamente aplicáveis ao ecossistema blockchain. A segurança em ambos os domínios depende fundamentalmente da correta gestão do ciclo de vida das dependências, desde sua definição até sua implementação e manutenção.

2.1.1 ***Injeção por Construtor***

Na injeção por construtor, as dependências são fornecidas no momento da criação do objeto, geralmente por meio de parâmetros no construtor. Essa abordagem apresenta vantagens como:

- **Imutabilidade**: dependências marcadas como `final` não podem ser modificadas após a inicialização;
- **Definição explícita de dependências**: facilita a identificação de dependências críticas durante revisões de código;

- **Prevenção de `NullPointerException`**: garante que as dependências essenciais estejam sempre presentes.

Por outro lado, os principais riscos dessa abordagem incluem:

- **Dependências circulares**: podem causar falhas na inicialização se não forem bem gerenciadas;
- **Exposição acidental**: construtores públicos podem ser explorados por APIs externas, aumentando o risco de ataques, especialmente em frameworks que permitem serialização maliciosa.

Um caso real é a vulnerabilidade CVE-2011-2730 no framework Spring (REDHAT, 2011). Nessa falha, era possível explorar o processo de *data binding*, em que o framework mapeava dados de requisições HTTP para objetos Java. A ausência de restrições adequadas permitia que um atacante manipulasse parâmetros HTTP para alterar propriedades sensíveis dos objetos.

*2.1.2 **Injeção por Setter (ou por Métodos)***

Na injeção por *setter*, as dependências são atribuídas por métodos após a criação do objeto, permitindo alterações dinâmicas. Sua principal vantagem é:

- **Flexibilidade**: útil para dependências opcionais ou configuráveis em tempo de execução.

Entretanto, os riscos são mais evidentes:

- **Estado inconsistente**: objetos podem ser utilizados antes de suas dependências essenciais estarem devidamente injetadas;
- **Sobrescrita maliciosa**: métodos públicos *setter* podem ser explorados para injetar objetos maliciosos via polimorfismo.

Um caso notório foi a falha CVE-2017-9805 no Apache Struts 2 (SECURITY, 2017), onde havia uma desserialização insegura de XML. Usuários podiam enviar *payloads* XML que eram automaticamente convertidos em objetos Java sem validação adequada. Isso permitiu a execução remota de comandos arbitrários, resultando no famoso vazamento de dados da empresa Equifax, que expôs informações pessoais de 148 milhões de pessoas nos EUA (G1, 2017).

2.1.3 ***Injeção por Interface***

Nesse modelo, as dependências são definidas por interfaces, e a implementação concreta é injetada por meio de um contêiner de injeção de dependência. Suas principais vantagens são:

- **Baixo acoplamento**: favorece a modularidade e aderência aos princípios SOLID;
- **Alta testabilidade**: facilita a criação de testes com uso de *mocks* e *stubs*.

No entanto, também existem riscos:

- **Poluição de interfaces**: interfaces excessivamente genéricas podem omitir dependências críticas;
- **Injeção de implementações maliciosas**: se o contêiner permitir substituição dinâmica (por exemplo, via arquivos XML mal configurados), é possível injetar implementações não verificadas.

Esse problema foi relatado por desenvolvedores que utilizam o Spring (SECURITY, 2022). Em certos cenários, o framework permitia a injeção de qualquer classe que implementasse uma interface pública, sem validação rigorosa da origem da classe, o que abria margem para ataques se o ambiente estivesse mal configurado.

Em síntese, a injeção de dependência, embora poderosa, exige atenção rigorosa à segurança em sua aplicação. Quando mal utilizada, pode abrir brechas tanto em sistemas tradicionais quanto em arquiteturas emergentes como a blockchain. A adoção de boas práticas, como o uso de abstrações seguras, validações rigorosas e gestão cuidadosa do ciclo de vida das dependências, é essencial para garantir a robustez e a segurança de sistemas modernos.

2.2 Trabalhos relacionados

O estudo de Kuk, Milić & Denić (2020) investiga a relação entre vulnerabilidades em código orientado a objetos e alto acoplamento, demonstrando que classes derivadas herdam não apenas funcionalidades, mas também fragilidades de design, como baixo encapsulamento e violações aos princípios SOLID. Por exemplo, se uma classe-mãe apresenta falhas de segurança, suas subclasses tendem a replicá-las, a menos que sejam aplicadas práticas adequadas de refatoração. Os autores destacam que a adesão aos princípios SOLID pode mitigar tais riscos. Essa discussão é pertinente ao presente trabalho, uma vez que a má implementação da Injeção de Dependência (DI) frequentemente infringe o

princípio da Inversão de Dependência, exacerbando o acoplamento e, consequentemente, as vulnerabilidades descritas no estudo.

Paralelamente, diversos estudos vêm sendo conduzidos com o objetivo de categorizar, detectar e mitigar vulnerabilidades em contratos inteligentes na plataforma Ethereum. Atzei, Bartoletti & Cimoli (2017) foram pioneiros ao propor uma taxonomia de falhas em contratos, agrupando-as em categorias como problemas relacionados a chamadas externas e manipulação de exceções. Tikhomirov (2017) realizaram uma avaliação sistemática de ferramentas de análise estática, como Mythril, Oyente e SmartCheck, comparando sua eficácia na detecção de vulnerabilidades. Já Torres *et al.* (2021) diferenciam-se ao propor uma abordagem baseada em aprendizagem de máquina para prever vulnerabilidades a partir do bytecode de contratos inteligentes.

Apesar da riqueza dessas contribuições, observa-se uma lacuna quanto à análise integrada entre vulnerabilidades de injeção de dependência e os riscos presentes em contratos inteligentes. Poucos estudos abordam explicitamente como práticas de desenvolvimento seguro aplicadas em sistemas orientados a objetos podem ser adaptadas para o contexto da blockchain. Assim, o presente trabalho busca justamente preencher esse vazio, propondo um olhar unificado sobre ambos os domínios.

3 METODOLOGIA

Este artigo adota uma abordagem qualitativa, com foco exploratório e descritivo, visando compreender a relação entre vulnerabilidades de segurança em software e métricas de análise orientadas a objetos. A metodologia foi estruturada em três atividades principais:

3.1 Atividade 1: Revisão sistemática da literatura

Inicialmente, realizou-se uma revisão de publicações acadêmicas recentes e relevantes sobre métricas de software e sua aplicação na predição ou explicação de vulnerabilidades. Foram selecionados artigos de periódicos indexados e conferências, como *IEEE Access*, *Software Mining* e *IJSEA*, garantindo a qualidade das fontes utilizadas. Essa revisão teve como objetivo identificar padrões e abordagens utilizadas na análise de segurança em projetos orientados a objetos.

3.2 Atividade 2: Análise de vulnerabilidades reais (estudo de caso)

Em seguida, foram analisadas vulnerabilidades reais documentadas em bancos de dados oficiais, como o *CVE (Common Vulnerabilities and Exposures)*, com foco em falhas amplamente divulgadas, como a `CVE-2022-22965` (SECURITY, 2022), `CVE-2017-9805` (SECURITY, 2017), entre outras. Casos como o ataque à Equifax em 2017 foram utilizados como estudo de caso para contextualizar a gravidade dessas vulnerabilidades e demonstrar a importância da detecção precoce via análise de código.

3.3 Atividade 3: Discussão e correlação com métricas de software

A terceira etapa consistiu na análise crítica dos casos à luz das métricas de software encontradas na literatura. Trabalhos como os de Sultana & Williams (2017) , Kuk, Milić & Denić (2020) e Kushwaha *et al.* (2022) forneceram base teórica para interpretar como características como acoplamento excessivo, baixa coesão e complexidade ciclomática elevada podem contribuir para o surgimento de falhas. Essa correlação foi feita de forma qualitativa, com o intuito de evidenciar padrões e reforçar a necessidade de práticas seguras no desenvolvimento orientado a objetos.

4 CRONOGRAMA

O cronograma a seguir (Tabela 1) apresenta a organização das atividades realizadas para o desenvolvimento deste trabalho, ao longo das quinzenas dos meses de maio e de junho.

Tabela 1 – Cronograma de atividades — Maio e Junho

Atividade	**Maio (1ª quinzena)**	**Junho (1ª quinzena)**	**Junho (2ª quinzena)**
Levantamento bibliográfico inicial e seleção de CVEs	X		
Leitura e análise crítica das fontes selecionadas	X		
Redação da revisão bibliográfica e dos trabalhos relacionados	X		
Envio e aplicação de correções da orientadora		X	
Redação da seção de metodologia		X	
Escrita da introdução		X	
Revisão geral e finalização do artigo			X

REFERÊNCIAS

ATZEI, N.; BARTOLETTI, M.; CIMOLI, T. A survey of attacks on ethereum smart contracts (sok). In: **Proceedings of the International Conference on Principles of Security and Trust (POST)**. [S.l.]: Springer, 2017. (LNCS, v. 10204), p. 164–186. Citado na página 10.

BUTERIN, V. **Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform**. 2014. <https://ethereum.org/en/whitepaper/>. White Paper. Citado na página 6.

G1. **Equifax, empresa de crédito dos EUA, sofre ataque hacker e dados de 143 milhões de pessoas são expostos**. 2017. Acessado em 20 maio 2025. Disponível em: <https://g1.globo.com/tecnologia/noticia/equifax-empresa-de-credito-dos-eua-sofre-ataque-hacker-e-dados-de-143-milhoes-de-pessoas-sao-expos
ghtml>. Citado na página 8.

HERN, A. Someone just stole $300m – how a bug in a popular cryptocurrency wallet opened the door to thieves. **The Guardian**, November 2017. Accessed: 2025-05-27. Disponível em: <https://www.theguardian.com/technology/2017/nov/08/cryptocurrency-300m-dollars-stolen-bug-ether>. Citado na página 7.

KUK, K.; MILIć, P.; DENIć, S. Object-oriented software metrics in software code vulnerability analysis. In: **2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)**. [S.l.: s.n.], 2020. p. 1–6. Citado 2 vezes nas páginas 9 e 11.

KUSHWAHA, S. S. *et al.* Systematic review of security vulnerabilities in ethereum blockchain smart contract. **IEEE Access**, v. 10, p. 6605–6621, 2022. Citado 2 vezes nas páginas 6 e 11.

NAKAMOTO, S. **Bitcoin: A Peer-to-Peer Electronic Cash System**. 2008. <https://bitcoin.org/bitcoin.pdf>. Citado na página 6.

REDHAT. **CVE-2011-2730: Red Hat Security Advisory**. 2011. <https://access.redhat.com/security/cve/cve-2011-2730>. Red Hat, Inc. Citado na página 8.

SECURITY, C. **CVE-2017-9805: Apache Struts REST Plugin with XStream Remote Code Execution**. 2017. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-9805>. MITRE Corporation. Citado 4 vezes nas páginas 3, 4, 8 e 11.

SECURITY, C. **CVE-2022-22965: Spring Framework RCE via Data Binding on JDK 9+**. 2022. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-22965>. MITRE Corporation. Citado 3 vezes nas páginas 3, 9 e 11.

SULTANA, K. Z.; WILLIAMS, B. J. Evaluating micro patterns and software metrics in vulnerability prediction. In: **2017 6th International Workshop on Software Mining (SoftwareMining)**. [S.l.: s.n.], 2017. p. 40–47. Citado na página 11.

TIKHOMIROV, S. Ethereum: State of knowledge and research perspectives. In: **Foundations and Practice of Security**. Springer International Publishing, 2017. (Lecture Notes in Computer Science, v. 10723), p. 206–221. Presented at the 10th International Symposium on Foundations & Practice of Security (FPS 2017), Nancy, France, October 23–25, 2017. Disponível em: <https://www.researchgate.net/publication/323234138_Ethereum_State_of_Knowledge_and_Research_Perspectives>. Citado na página 10.

TORRES, P. E. *et al.* A systematic review of physical–digital play technology and developmentally relevant child behaviour. **International Journal of Child-Computer Interaction**, Elsevier, v. 30, p. 100323, 2021. Accessed: 2025-05-27. Disponível em: <https://doi.org/10.1016/j.ijcci.2021.100323>. Citado na página 10.