Um Framework Unificado para Auditoria de Contratos Inteligentes com Análise Multicamada

Guilherme A. Soares[1], **João L.D.S Filho** [2], **Nicholas P. Fontanini**[3], **Bruno Evaristo** [4]

[1] Universidade Estadual do Oeste do Paraná (Unioeste)
Cascavel – PR – Brazil

[2]Universidade Federal do Ceará (UFC)
Itapajé – CE – Brazil

[3]Universidade Estadual de Campinas (Unicamp)
Campinas – SP – Brazil

[4]Centro de Pesquisa e Desenvolvimento em Telecomunicações (CPQD)
Campinas – SP – Brazil

altmeyergui,joaofilho1467, nicholas.p.fontanini}@gmail.com,

elderb@cpqd.com.br

Abstract. *Smart contracts manage high-value digital assets, but security flaws frequently result in irreversible financial losses. Although several automated auditing tools exist, their isolated execution often yields high rates of false positives and false negatives. This paper proposes and evaluates a unified framework for auditing Solidity smart contracts, combining static analysis (Slither), symbolic execution (Mythril), and dynamic testing (Foundry). The architecture orchestrates tool execution, unifies heterogeneous outputs using SARIF, and uses a Large Language Model (LLM) only in the post-processing stage to generate explainable reports. Evaluated on a curated dataset of 53 contracts from SmartBugs, the framework achieved an F1-Score of 92.93%, outperforming Slither (72.28%) and Mythril (88.42%).*

Resumo. *Contratos inteligentes gerenciam ativos digitais de alto valor, mas falhas de segurança frequentemente causam perdas financeiras irreversíveis. Este trabalho propõe e avalia um framework unificado para auditoria de contratos em Solidity, orquestrando análise estática (Slither), execução simbólica (Mythril) e testes dinâmicos (Foundry). A arquitetura normaliza saídas heterogêneas em SARIF e emprega LLM no pós-processamento para produzir relatórios explicáveis. Em um dataset de 53 contratos do SmartBugs, o framework atingiu F1-Score de 92,93%, superando Slither (72,28%) e Mythril (88,42%).*

1. Introdução

Blockchain consolidou-se como infraestrutura para aplicações descentralizadas e contratos inteligentes [Nakamoto 2008, Zhang et al. 2025]. Nesse contexto, falhas de segurança podem gerar perdas financeiras severas e irreversíveis, como evidenciado por incidentes históricos de mercado [U.S. Securities and Exchange Commission 2017]. Embora existam ferramentas robustas de auditoria, sua execução isolada tende a produzir lacunas de

cobertura e ruído analítico [Ferreira et al. 2020]. Em geral, análise estática prioriza velocidade, execução simbólica amplia profundidade semântica e validação dinâmica ajuda a confirmar explorabilidade.

Este trabalho apresenta o OrchestralSec, um ambiente unificado para auditoria de segurança em Solidity que combina análise estática, execução simbólica e *fuzzing*. A proposta integra resultados em um modelo comum, prioriza achados por uma taxonomia comum e acrescenta explicabilidade orientada por LLM para reduzir esforço de triagem humana. O foco do artefato não é introduzir um novo detector de vulnerabilidades, mas organizar ferramentas complementares e tornar a análise mais reprodutível e acionável.

2. Fundamentação Teórica

2.1. Conceitos e vulnerabilidades-alvo

Contratos inteligentes são programas autoexecutáveis com regras imutáveis após implantação, o que aumenta o impacto de falhas de implementação [Li et al. 2022, Alchini 2025]. Entre as classes de vulnerabilidade avaliadas neste estudo, destacam-se: *access control*, reentrância, erros aritméticos, *front-running*, *time manipulation*, *bad randomness*, DoS e chamadas de baixo nível não verificadas.

2.2. Ferramentas de análise

A abordagem combina: (i) Slither para inspeção estrutural de código; (ii) Mythril para execução simbólica e exploração de caminhos críticos na EVM; e (iii) Foundry/Forge para validação dinâmica por *fuzzing* e invariantes [Feist et al. 2019, Mueller 2018, Foundry Contributors 2024]. Essa composição aumenta a cobertura de detecção e reduz dependência de uma única técnica.

3. Trabalhos Relacionados

SmartBugs consolidou a avaliação comparativa de ferramentas de auditoria em larga escala, porém estudos empíricos mostram persistência de falsos positivos e limitações de cobertura quando motores são usados isoladamente [Ferreira et al. 2020, Durieux et al. 2020]. Em abordagens híbridas recentes, Slither costuma ser combinado com camadas de interface ou IA generativa, enquanto validação dinâmica permanece menos explorada [Carrera et al. 2025]. Soluções baseadas diretamente em LLMs melhoram explicação textual, mas ainda carecem de validação prática [Chen et al. 2025].

Diferentemente dessas abordagens, o OrchestralSec integra detecção multicamada e validação dinâmica no mesmo fluxo, com normalização de achados e priorização contextual para suportar decisões de correção. Assim, sua principal contribuição está na integração prática e na redução do esforço analítico do auditor, e não na proposição de uma nova técnica de detecção.

4. Metodologia

4.1. Pipeline e orquestração

A auditoria utiliza execução coordenada e predominantemente paralela: o contrato é preparado uma única vez e então analisado por Slither, Mythril e Foundry, cujas saídas convergem na etapa de agregação. Os três motores foram escolhidos por representarem,

respectivamente, análise estática sobre código-fonte, execução simbólica sobre bytecode e validação dinâmica por *fuzzing*/invariantes, além de serem ferramentas consolidadas e abertas [Feist et al. 2019, Mueller 2018, Foundry Contributors 2024]. O protótipo atual emprega *containers* Docker com versões controladas de *solc*, mas essa decisão é de implementação. Para análise em escala, o módulo dinâmico aplica *invariant fuzzing* com propriedades genéricas de segurança.

4.2. Normalização e explicabilidade

Os resultados são convertidos para SARIF e mapeados ao SWC Registry, permitindo consolidar alertas equivalentes de motores distintos [Standard 2020, Vogelgesang et al. 2020]. Embora a SWC não receba atualizações frequentes desde 2020, ela foi empregada neste trabalho como taxonomia estável de interoperabilidade entre ferramentas, e não como fonte exclusiva de conhecimento. Na prática, o framework preserva também o rótulo original emitido por cada motor e permite revisão manual quando há divergência taxonômica.

Para combinar os resultados, o OrchestralSec aplica três heurísticas simples: (i) normalização de achados semanticamente equivalentes para um mesmo identificador; (ii) agregação por união, preservando todos os alertas e elevando a confiança quando há concordância entre motores independentes; e (iii) reforço de confiança quando o comportamento suspeito pode ser corroborado por evidências dinâmicas do Foundry. Quando há divergência entre ferramentas, o alerta é mantido com a proveniência original e confiança menor, em vez de ser descartado. Assim, o framework não combina apenas verdadeiros positivos: ele agrega todos os alertas produzidos e posteriormente os classifica para avaliação. A arquitetura também é extensível: novos motores podem ser acoplados desde que forneçam adaptadores para o esquema comum em SARIF.

Na etapa experimental, os rótulos de verdadeiro positivo, falso positivo e falso negativo foram obtidos por comparação com a anotação de referência do dataset SmartBugs e por verificação manual dos casos ambíguos. Os falsos negativos não são filtrados pelo sistema; ao contrário, são contabilizados sempre que uma vulnerabilidade anotada não aparece no relatório unificado. A camada de LLM atua somente na explicação textual dos achados [Roziere et al. 2023]. No protótipo descrito, utilizou-se uma LLM da família Gemini via API, mas a arquitetura admite outros modelos compatíveis; o *prompt* recebe trecho vulnerável, origem, categoria, evidências e a instrução para produzir resumo, impacto e mitigação.

5. Avaliação Experimental

5.1. Configuração e métricas

A avaliação utilizou 53 contratos do SmartBugs, cobrindo múltiplas categorias SWC. A seleção foi intencional, buscando diversidade de classes com rótulo conhecido e viabilidade de inspeção manual dos alertas consolidados. Portanto, o estudo deve ser interpretado como uma avaliação empírica controlada, adequada para um artefato de workshop, mas insuficiente para sustentar generalizações amplas sobre contratos em produção. As métricas principais foram Precisão (P), *Recall* (R) e F1-Score, calculadas por:

$$P = \frac{VP}{VP + FP}, \quad R = \frac{VP}{VP + FN}, \quad F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{1}$$

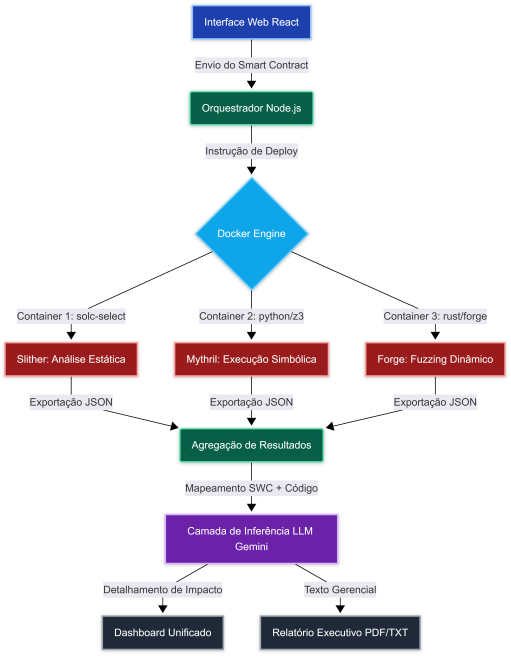

Figura 1. Arquitetura de orquestração do OrchestralSec.

Tabela 1. Desempenho consolidado das abordagens avaliadas (53 contratos).

Abordagem	VP	FP	FN	Precisão (%)	Recall (%)	F1 (%)
Slither	30	3	20	90,90	60,00	72,28
Mythril	42	5	6	89,36	87,50	88,42
OrchestralSec	46	4	3	92,00	93,88	92,93

Nota: VP = Verdadeiros Positivos, FP = Falsos Positivos e FN = Falsos Negativos.

O *baseline* deste estudo compara o relatório agregado do framework com seus principais motores internos, o que mostra ganho incremental de cobertura, mas não substitui comparações com outros orquestradores. Essa comparação mais ampla é uma limitação do escopo atual.

A Tabela 1 evidencia que o OrchestralSec obteve o melhor equilíbrio entre cobertura e assertividade. Em relação ao Slither, o ganho de *recall* foi de 33,88 pontos percentuais (de 60,00% para 93,88%), indicando redução substancial de vulnerabilidades não detectadas (FN). Esse resultado é particularmente relevante para auditorias de segurança, nas quais a omissão de falhas críticas tende a gerar maior impacto que alertas adicionais.

Em comparação ao Mythril, o framework também apresentou avanço consistente: aumento de 4,52 pontos no *recall* (87,50% para 93,88%), elevação de 2,64 pontos na precisão (89,36% para 92,00%) e crescimento de 4,51 pontos no F1-Score (88,42% para 92,93%). Esses ganhos mostram que a estratégia híbrida não apenas amplia a cobertura, mas preserva a qualidade dos achados, reduzindo ruído operacional na triagem.

Sob uma perspectiva prática, a combinação entre Slither, Mythril e validação dinâmica no Foundry melhora a confiabilidade da decisão de correção. A análise estática contribui com velocidade e padronização, a execução simbólica amplia profundidade se-

mântica e o módulo dinâmico ajuda a confirmar cenários de exploração. Como efeito, o auditor recebe um conjunto de evidências mais robusto, priorizado e acionável, reduzindo retrabalho na inspeção manual.

Algumas categorias, contudo, permaneceram desafiadoras. O *recall* de 60% observado para manipulação de tempo indica que 40% dos casos anotados dessa classe ainda não foram recuperados pelo sistema, o que não é um resultado ideal para um cenário de auditoria crítica. Esse comportamento sugere limitações estruturais das técnicas empregadas para detectar dependências sutis de `block.timestamp`, especialmente quando a vulnerabilidade depende de contexto de execução ou de lógica de negócio mais específica.

5.2. Limitações e ameaças à validade

Os resultados obtidos não implicam aumento direto da segurança dos contratos, mas sim melhora no suporte à detecção, consolidação e interpretação dos achados. A camada de LLM, por exemplo, agrega valor à comunicação dos resultados, porém não contribui diretamente para descobrir novas vulnerabilidades. Além disso, o uso de um repositório conhecido e de uma base pequena tende a produzir um cenário mais controlado do que contratos reais, que frequentemente exibem maior diversidade arquitetural, integrações externas e mecanismos de evasão.

O framework também herda limitações dos motores subjacentes: vulnerabilidades fora do escopo dos detectores empregados, contratos altamente ofuscados, dependências externas complexas e padrões emergentes podem passar despercebidos. Esses fatores reforçam que o OrchestralSec deve ser entendido como um apoio incremental à auditoria, e não como substituto para revisão especializada.

6. Conclusão e Trabalhos Futuros

O OrchestralSec demonstrou que a integração entre Slither, Mythril, Foundry e uma camada de explicabilidade por LLM produz ganhos quantitativos e qualitativos no suporte à auditoria de contratos inteligentes. Em comparação com abordagens isoladas, o framework elevou o F1-Score para 92,93% e reduziu esforço de triagem manual. A contribuição observada é incremental, mas relevante do ponto de vista prático: organizar achados heterogêneos, aumentar cobertura empírica e oferecer relatórios mais acionáveis.

Como continuidade, destacam-se: geração automática de provas de conceito, validação de correções por testes de regressão, integração em pipelines CI/CD e ampliação para técnicas de verificação formal.

Agradecimentos

Os autores agradecem o apoio concedido pelo Ministerio da Ciencia, Tecnologia e Inovação (MCTI) por meio de recursos da Lei no 8.248, de 23 de outubro de 1991, no âmbito do PPI SOFTEX (coordenado pela Softex e publicado como Residência em TIC 11, DOU 01245.011733/2022-83).

Referências

Alchini, C. A. (2025). Análise de ameaças e vulnerabilidades em blockchains permissionadas. Trabalho de conclusão de curso (graduação), Universidade Federal de Santa Catarina, Florianópolis.

Carrera, L., Cordeiro, R., and Abelém, A. (2025). Auditai: Automatizando e facilitando a auditoria de contratos inteligentes com relatórios contextuais gerados por ia. In *Anais do VII Workshop em Blockchain: Teoria, Tecnologias e Aplicações*, pages 140–153, Porto Alegre, RS, Brasil. SBC.

Chen, C., Su, J., Chen, J., Wang, Y., Bi, T., Yu, J., Wang, Y., Lin, X., Chen, T., and Zheng, Z. (2025). When chatgpt meets smart contract vulnerability detection: How far are we? *ACM Transactions on Software Engineering and Methodology*, 34(4):100.

Durieux, T., Ferreira, J. F., Abreu, R., and Cruz, P. (2020). Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE '20)*, pages 530–541. ACM.

Feist, J., Grieco, G., and Groce, A. (2019). Slither: A static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 8–15. IEEE.

Ferreira, J. F., Cruz, P., Durieux, T., and Abreu, R. (2020). Smartbugs: A framework to analyze solidity smart contracts. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1349–1352.

Foundry Contributors (2024). Foundry: Blazing fast, portable and modular toolkit for ethereum application development. `https://github.com/foundry-rs/foundry`. Acessado em: 05 fev. 2026.

Li, P., Li, S., Ding, M., Yu, J., Zhang, H., Zhou, X., and Li, J. (2022). A vulnerability detection framework for hyperledger fabric smart contracts based on dynamic and static analysis. In *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering (EASE '22)*, pages 366–374, New York, NY, USA. ACM.

Mueller, B. (2018). Mythril: Security analysis tool for ethereum smart contracts. `https://github.com/ConsenSys/mythril`.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Acessado em: 28 jan. 2026.

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al. (2023). Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Standard, O. (2020). Static analysis results interchange format (sarif) version 2.1.0. Technical report, OASIS Open.

U.S. Securities and Exchange Commission (2017). Report of investigation pursuant to section 21(a) of the securities exchange act of 1934: The dao. `https://www.sec.gov/litigation/investreport/34-81207.pdf`. Release No. 81207. Acessado em: 28 jan. 2026.

Vogelgesang, T. et al. (2020). Smart contract weakness classification and test cases. *IEEE Standard for Smart Contract Security*.

Zhang, C., Dou, F., and Li, X. (2025). Dos attacks and defense technologies in blockchain systems: A hierarchical analysis. arXiv preprint arXiv:2507.22611.