OrchestralSec: Um Framework Híbrido e Explicável para Segurança de Contratos Inteligentes Solidity

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,micholas.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 the SARIF standard, and leverages a Large Language Model (LLM) to translate raw logs into contextual, explainable reports. Evaluated on a curated dataset of 53 contracts from the SmartBugs repository, the proposed framework achieved an F1-Score of 92.93%, substantially outperforming the isolated performance of Slither (72.28%) and Mythril (88.42%). The results demonstrate that hybrid orchestration mitigates the structural limitations of individual engines, reduces the auditor's cognitive load, and consolidates a robust platform for secure development in the Web3 ecosystem.*

Resumo. *Contratos inteligentes gerenciam ativos digitais de alto valor, mas falhas de segurança frequentemente causam perdas financeiras irreversíveis. Embora existam diversas ferramentas de auditoria automatizada, seu uso isolado gera altas taxas de falsos positivos e falsos negativos. Este trabalho propõe e avalia um framework unificado para auditoria de contratos inteligentes em Solidity, orquestrando análise estática (Slither), execução simbólica (Mythril) e testes dinâmicos (Foundry). A arquitetura unifica os resultados heterogêneos utilizando o padrão SARIF e aplica um Modelo de Linguagem de Grande Escala (LLM) para traduzir logs brutos em relatórios contextuais explicáveis. Avaliado em um dataset curado de 53 contratos do repositório SmartBugs, o framework alcançou um F1-Score de 92,93%, superando substancialmente o desempenho*

isolado do Slither (72,28%) e do Mythril (88,42%). Os resultados demonstram que a orquestração híbrida mitiga as limitações estruturais de cada motor, reduz a carga cognitiva do auditor e consolida-se como uma plataforma robusta e eficaz para o desenvolvimento seguro no ecossistema Web3.

1. Introdução

Blockchain constitui um novo paradigma de aplicação computacional, introduzido pela primeira vez por Satoshi Nakamoto em 2008 [Nakamoto 2008], o qual possibilita a integração de armazenamento distribuído, comunicação ponto a ponto (P2P), mecanismos de consenso e técnicas criptográficas.

A tecnologia tem apresentado avanços substanciais no cenário internacional, consolidando-se como infraestrutura essencial em múltiplos setores [Zhang et al. 2025]. Em contratos inteligentes, o uso de blockchain incrementa a eficiência operacional e eleva os padrões de segurança; em cadeias de suprimentos, viabiliza maior transparência e rastreabilidade de produtos e processos; no setor financeiro, otimiza transações de pagamentos transfronteiriços e a gestão de ativos; e, na área da saúde, estabelece uma arquitetura robusta para o compartilhamento seguro de dados clínicos. Não obstante, a rápida expansão e a intensificação do uso dessa tecnologia geram preocupações significativas relacionadas à sua estabilidade, segurança e resiliência a ameaças emergentes.

Com o aumento da complexidade de cenários de aplicação, novos desafios voltados á segurança dessas aplicações começaram a surgir. Diferentemente do software convencional, as vulnerabilidades em contratos inteligentes tendem a gerar prejuízos financeiros severos quando exploradas por atacantes [Zhang et al. 2025]. Como exemplo histórico temos o ataque DAO realizado em 2016, no qual resultou em uma perda de aproximadamente 60 milhões de dólares em ether [U.S. Securities and Exchange Commission 2017]. Evidencia-se, assim, que os contratos inteligentes se tornaram o ponto mais crítico na segurança de sistemas blockchain.

A comunidade científica tem envidado esforços significativos no desenvolvimento de ferramentas de análise automatizada capazes de identificar e mitigar vulnerabilidades em contratos inteligentes [Ferreira et al. 2020]. Entretanto, a ampla variedade de ferramentas disponíveis dificulta a seleção daquelas mais adequadas a cada cenário de uso, bem como a correta configuração do ambiente de execução específico requerido por cada solução.

O objetivo deste trabalho consiste no desenvolvimento e validação de um ambiente unificado para auditoria de segurança de contratos inteligentes. A metodologia proposta integra a execução coordenada de ferramentas consolidadas de análise estática, tais como Mythril [ConsenSys Diligence 2024] e Slither [Feist et al. 2019], com a robustez de testes conduzidos por meio do framework Foundry [Foundry Contributors 2024], aprofundando a investigação de segurança por meio de testes automatizados. Ao confrontar os resultados da análise estática com simulações de execução em condições próximas às reais e com dados empíricos de consumo de gás, a metodologia proposta busca reduzir a incidência de falsos positivos e, consequentemente, aumentar a robustez e a confiabilidade dos diagnósticos finais.

2. Fundamentação Teórica

2.1. Blockchain

Segundo [Braga et al. 2017], blockchain é uma base de dados compartilhada pelos nós de um sistema distribuído organizado em uma rede *peer-to-peer* ($P2P$). Qualquer nó com acesso específico pode consultar e modificar essa base, sendo sua integridade definida pela distribuição de registros dos nós participantes, os registros dessa base são comumente chamado de blocos, blocos podem apenas ser adicionados e assim que entram na base não podem ser excluídos ou modificados, guardando a história desde sua criação até o presente momento.

2.2. Smart Contracts

Contratos Inteligentes são programas autoexecutáveis caracterizados pela autoverificação e pela ausência de intermediários [Li et al. 2022], que vêm se popularizando bastante junto com a tecnologia blockchain. Sua implementação em blockchain viabiliza a transferência eficiente de valores, a gestão de ativos e a troca de informações [Li et al. 2022]. Seu processo de execução também impede que qualquer uma das partes repudie ou interrompa o contrato, assegurando sua efetivação sem a necessidade de confiança mútua prévia.

Entretanto, a natureza imutável dos contratos inteligentes e a gestão direta de ativos financeiros tornam a segurança desses artefatos um aspecto crítico. Diferentemente de softwares tradicionais, falhas em contratos inteligentes são de difícil correção após a implantação e podem resultar em perdas financeiras irreversíveis [Alchini 2025]. Apesar da crescente atenção dedicada a essas vulnerabilidades, observa-se que elas tendem a se recriar de forma recorrente: a cada ano, novos ataques são identificados ou vetores já conhecidos são modificados e reutilizados.

2.3. Vulnerabilidades Comuns

Controle de Acesso (*Access Control*): Falhas na restrição de funções críticas do contrato. Ocorre frequentemente pela ausência ou má configuração de modificadores de visibilidade (como permitir que qualquer usuário chame uma função de destruição ou saque), lógica falha de verificação de permissões, ou o uso inseguro do `tx.origin` no lugar de `msg.sender` para autenticação, viabilizando ataques de *phishing*.

Erros Aritméticos (*Integer Overflow* e *Underflow*): Manifesta-se quando uma operação aritmética ultrapassa os limites de armazenamento definidos pelo tipo de variável (por exemplo, `uint256`), provocando o reinício cíclico do valor e, consequentemente, comprometendo a consistência e a correção da lógica contábil implementada. É particularmente crítico em contratos compilados em versões do Solidity anteriores à 0.8.0.

Reentrância (*Re-entrancy*): Vulnerabilidade crítica em que um contrato externo malicioso executa chamadas recursivas a uma função antes da atualização do estado interno do contrato original (por exemplo, a dedução do saldo do usuário), possibilitando a extração repetida e indevida de fundos em um único ciclo de transação.

Dependência de Ordem de Transação (*Front-Running*): Manifesta-se quando o estado final do contrato inteligente é sensível à ordem em que as transações são

incluídas em blocos e mineradas. Agentes maliciosos monitoram transações pendentes no *mempool* e submetem operações concorrentes com taxas de *gas* mais elevadas, obtendo prioridade de execução para explorar situações economicamente vantajosas.

Negação de Serviço (*Denial of Service - DoS*): Ocorre quando um atacante consegue interromper, travar ou encarecer excessivamente o funcionamento de um contrato inteligente. Pode ser executado forçando laços de repetição infinitos que esgotam o limite de gás (*gas limit*) ou manipulando variáveis de estado de forma que chamadas subsequentes de usuários legítimos falhem permanentemente.

Manipulação de Tempo (*Time Manipulation*): Uso não seguro da variável `block.timestamp` como condição para a execução de lógicas de negócio cruciais, como a liberação de fundos. Como os nós validadores possuem uma margem de tolerância para definir o registro de data e hora do bloco, torna-se possível a criação de cenários artificialmente favoráveis para burlar restrições temporais.

Aleatoriedade Insegura (*Bad Randomness*): Tentativa de gerar valores pseudoaleatórios utilizando variáveis públicas e determinísticas da blockchain, como o *timestamp* ou o *hash* de blocos anteriores. Devido à transparência do *ledger*, esses valores podem ser previstos ou levemente manipulados por mineradores, comprometendo sistemas de loterias ou sorteios *on-chain*.

Chamadas de Baixo Nível Não Verificadas (*Unchecked Low-Level Calls*): Omissão na verificação do valor de retorno booleano em instruções diretas de comunicação externa, como `call`, `delegatecall` ou `send`. O contrato infere indevidamente que a transferência ou execução foi bem-sucedida, o que pode resultar em graves inconsistências no estado contábil caso a chamada falhe silenciosamente.

Endereços Curtos (*Short Addresses*): Vulnerabilidade de infraestrutura e formatação onde dados de entrada menores que o padrão esperado (como um endereço incompleto) fazem com que a *Ethereum Virtual Machine* (EVM) preencha os *bytes* faltantes com zeros no final da estrutura de dados (*payload*). Isso desloca os parâmetros da função, podendo multiplicar acidentalmente e de forma maliciosa o valor de uma transferência de *tokens*.

2.4. Ferramentas de Análise

A complexidade inerente aos contratos inteligentes, somada à irreversibilidade de suas falhas, impulsionou o desenvolvimento de ferramentas especializadas em auditoria automatizada. O objetivo central dessas soluções é identificar vulnerabilidades precocemente, mitigando riscos de exploração antes do deployment na rede principal. Para isso, o ecossistema divide-se majoritariamente entre a análise estática, que examina o código-fonte sem executá-lo, e a execução simbólica, que explora matematicamente os estados do contrato para identificar fluxos lógicos perigosos.

Como expoente da análise estática, o Slither consolida-se como uma das ferramentas mais proeminentes na literatura. Sua operação baseia-se na conversão do código Solidity para uma representação intermediária denominada SlithIR. Esse processo facilita a detecção de padrões de erro conhecidos, como reentrância e falhas de visibilidade. Por atuar diretamente no código-fonte, o Slither oferece alta performance e baixa taxa de falsos negativos em verificações de conformidade, sendo peça-chave em pipelines de integração contínua (CI/CD) para garantir a higiene do código durante o desenvolvimento.

Complementarmente à abordagem estrutural, o Mythril foca na análise de segurança do bytecode da Ethereum Virtual Machine (EVM). Ao empregar técnicas de execução simbólica e análise de mancha (taint analysis), a ferramenta consegue detectar vulnerabilidades complexas que dependem do estado da blockchain, simulando interações maliciosas reais. Enquanto o Slither prioriza a estrutura sintática, o Mythril mergulha na semântica da execução, identificando vetores de ataque que passariam despercebidos em análises superficiais, o que o torna um pilar essencial para a segurança profunda de artefatos digitais [Josue N. Campos 2025].

Diferenciando-se das abordagens puramente estáticas ou simbólicas, o ConFuzzius introduz a técnica de fuzzing híbrido para a análise de contratos na EVM. Esta ferramenta combina a geração aleatória de entradas (fuzzing) com a precisão da execução simbólica para maximizar a cobertura de código. Essa integração permite que o analisador alcance estados profundos e caminhos de execução complexos, muitas vezes inacessíveis por métodos tradicionais. Ao incorporar um componente evolutivo que aprende com execuções prévias, o ConFuzzius entrega alta eficiência e baixíssima taxa de falsos positivos, visto que cada falha detectada é acompanhada por um caso de teste concreto que comprova a vulnerabilidade em tempo de execução [Josue N. Campos 2025].

3. Trabalhos Relacionados

A contínua evolução de projetos baseados em blockchain, tanto no setor público quanto no setor privado, tem impulsionado o desenvolvimento de ferramentas especializadas para a auditoria de contratos inteligentes [Zhang et al. 2025]. Nesse contexto, o framework Smart-Bugs destacou-se como uma solução inovadora ao adotar uma arquitetura modular, com integração a diversas ferramentas de análise [Ferreira et al. 2020]. Em 2026, o projeto passou a oferecer suporte a 25 ferramentas de análise acessíveis por meio de interface de linha de comando.

Entretanto, conforme apontado por Durieux et al. [Durieux et al. 2020], a aplicação do Smart-Bugs a um conjunto de 47.518 contratos evidenciou a ocorrência de um número substancial de falsos positivos. Com o objetivo de mitigar essa limitação, Carrera et al. [Carrera et al. 2025] propuseram a integração da ferramenta de análise estática Slither com técnicas de inteligência artificial generativa (Groq AI), de modo a auxiliar na geração de relatórios e na inspeção de contratos a partir de uma interface gráfica e de um chatbox de suporte. Todavia, para análises mais complexas e para a consulta de métricas detalhadas relativas aos contratos, a ferramenta ainda não oferece suporte adequado.

Utilizando uma abordagem semelhante, Chen et al. [Chen et al. 2025] investigaram a eficácia dos Modelos de Linguagem de Grande Escala (LLMs), como o ChatGPT, na detecção direta de vulnerabilidades. O estudo concluiu que, embora os LLMs possuam notável capacidade de compreensão semântica para explicar trechos de código, eles ainda sofrem com altas taxas de falsos positivos e alucinações em fluxos de execução complexos, mostrando-se limitados como mecanismos primários de detecção. Ademais, para a comprovação da explorabilidade e o perfilamento detalhado de métricas operacionais, essas soluções ainda não oferecem suporte adequado.

Sob a ótica da avaliação de ferramentas automatizadas, Fonseca et al. [Fonseca et al. 2025] conduziram um estudo de caso recente focado especificamente na evolução temporal do Mythril e do Slither. Ao aplicar as versões mais recentes dessas

ferramentas em um conjunto de 69 contratos previamente catalogados, os resultados revelaram que, apesar de o Mythril ter diminuído a incidência de falsos positivos e o Slither ter melhorado a identificação de falhas de controle de acesso (*Access Control*), ambas as soluções ainda apresentam limitações significativas quando executadas de forma isolada.

Tendo isso em vista, este trabalho propõe a mitigação dessas limitações metodológicas por meio da orquestração do ecossistema Foundry em conjunto com as ferramentas Slither e Mythril. Diferentemente de abordagens que dependem exclusivamente de heurísticas estáticas ou predições por IA, o sistema resultante viabiliza a execução de testes de estresse empíricos (*fuzzing*) e o perfilamento detalhado do consumo de gás, além de disponibilizar uma interface de observabilidade avançada. Essa abordagem permite a validação do comportamento do contrato inteligente em ambiente isolado, utilizando dados reais de execução para corroborar ou refutar de maneira definitiva os alertas produzidos pelas camadas de análise anteriores.

4. Metodologia

O presente trabalho busca desenvolver uma plataforma integrada de análise de segurança em contratos inteligentes Solidity. A partir da criação de um artefato tecnológico voltado à solução de um problema prático — a complexidade e fragmentação das ferramentas de auditoria de *smart contracts* [Atzei et al. 2017].

A plataforma unifica técnicas de análise estática, execução simbólica e *fuzzing*, utilizando motores consolidados na literatura e na indústria, como Slither [Feist et al. 2019], Mythril [Mueller 2018] e Forge [Foundry Contributors 2024]. O diferencial reside na camada de interpretação inteligente baseada no *Smart Contract Weakness Classification* (SWC Registry) [**?**] e nas diretrizes de melhores práticas da ConsenSys.

4.1. Pipeline de Análise e Orquestração de Motores

Para garantir a eficiência e a cobertura da análise, o artefato implementa um pipeline de execução sequencial e paralela. A análise inicia-se com o motor Slither, devido ao seu baixo custo computacional e alta velocidade na detecção de vulnerabilidades sintáticas e de fluxo de dados [Feist et al. 2019]. Caso o contrato apresente uma estrutura válida, o sistema dispara simultaneamente os processos de Execução Simbólica (Mythril) e Fuzzing (Forge). Também é possível executar cada ferramenta individualmente, tirando um maior proveito principalmente da ferramenta foundry, que permite o auditor escrever testes para o contrato.

A orquestração desses motores é realizada através de *containers* isolados via Docker. Esta abordagem é crítica para a segurança e estabilidade do sistema, permitindo que cada análise ocorra em um ambiente com versões específicas do compilador Solidity (*solc*), evitando conflitos de dependências e garantindo a reprodutibilidade dos resultados [Vogelgesang et al. 2020].

Para viabilizar a análise em escala das 53 amostras, a execução do Foundry foi configurada para utilizar testes de invariantes (*invariant fuzzing*) padronizados. Em vez de testes unitários manuais para cada contrato, o orquestrador gera dinamicamente um conjunto de propriedades genéricas de segurança, como a ausência de travamento de saldo e a imutabilidade não autorizada de variáveis de estado críticas, submetendo o bytecode a milhares de transações aleatórias para identificar quebras de invariantes.

4.2. Normalização e Taxonomia de Vulnerabilidades

Um desafio crítico na integração de ferramentas heterogêneas é a disparidade de formatos de saída (*outputs*). Enquanto o Slither gera relatórios focados em detectores de AST (*Abstract Syntax Tree*), o Mythril reporta estados de execução simbólica em formato JSON proprietário. Esta fase da metodologia foca na construção de um Motor de Unificação de Dados.

A normalização é processada em três camadas:

- Ingestão e Parsing: Filtros específicos para cada ferramenta convertem os logs brutos em um esquema de dados comum baseado no padrão *Static Analysis Results Interchange Format* (SARIF) [Standard 2020].
- Mapeamento Taxonômico: Cada falha detectada é vinculada a um identificador único do *Smart Contract Weakness Classification* (SWC Registry). Isso permite que o usuário identifique, por exemplo, que o "Reentrancy"detectado pelo Slither e o "State Change After External Call"do Mythril referem-se ao mesmo risco estrutural (SWC-107) [Vogelgesang et al. 2020].
- Cálculo de Severidade Agregada: Implementa-se uma matriz de risco que pondera a confiança de cada ferramenta. Se múltiplos motores confirmam a mesma vulnerabilidade, o sistema eleva o nível de severidade e prioridade de correção.

4.3. Mecanismo de Explicabilidade e Integração com LLM

Para mitigar a lacuna de conhecimento entre auditores seniores e desenvolvedores iniciantes, o artefato propõe uma camada de Inteligência de Segurança Explicável. A metodologia descreve o uso de Modelos de Linguagem de Grande Escala (LLMs) não apenas como geradores de texto, mas como motores de raciocínio contextual [Roziere et al. 2023].

O processo de integração da camada de inteligência adota um fluxo de *Prompt Engineering* estruturado para a geração automatizada de relatórios:

1. **Agregação de Contexto:** O *backend* extrai o fragmento de código vulnerável (*snippet*), os rastros de execução fornecidos pelas ferramentas (como Slither e Mythril) e a definição técnica do SWC correspondente.
2. **Inferência Contextual:** Estes dados alimentam a LLM com uma instrução específica para traduzir a vulnerabilidade técnica bruta em um impacto de negócio tangível.
3. **Síntese do Relatório Executivo:** A LLM processa o contexto e estrutura um documento voltado para a tomada de decisão. O artefato gerado inclui um resumo gerencial, a descrição do impacto financeiro e reputacional, exemplos didáticos de cenários de ataque e recomendações técnicas para mitigação, facilitando o fluxo de correção pelas equipes de desenvolvimento.

A incorporação do Modelo de Linguagem não alterou a capacidade primária de detecção das ferramentas (Slither e Mythril), mas agregou valor substancial na etapa de triagem. A análise direta de logs gerados por execução simbólica exige um alto tempo de interpretação por parte do auditor.

Ao traduzir o padrão SARIF e os rastros de execução para uma linguagem natural estruturada, o módulo LLM reduziu drasticamente a carga cognitiva da auditoria. Tendo ganhos na geração de relatórios e ideias para serem exploradas no ecossistema integrado do foundry.

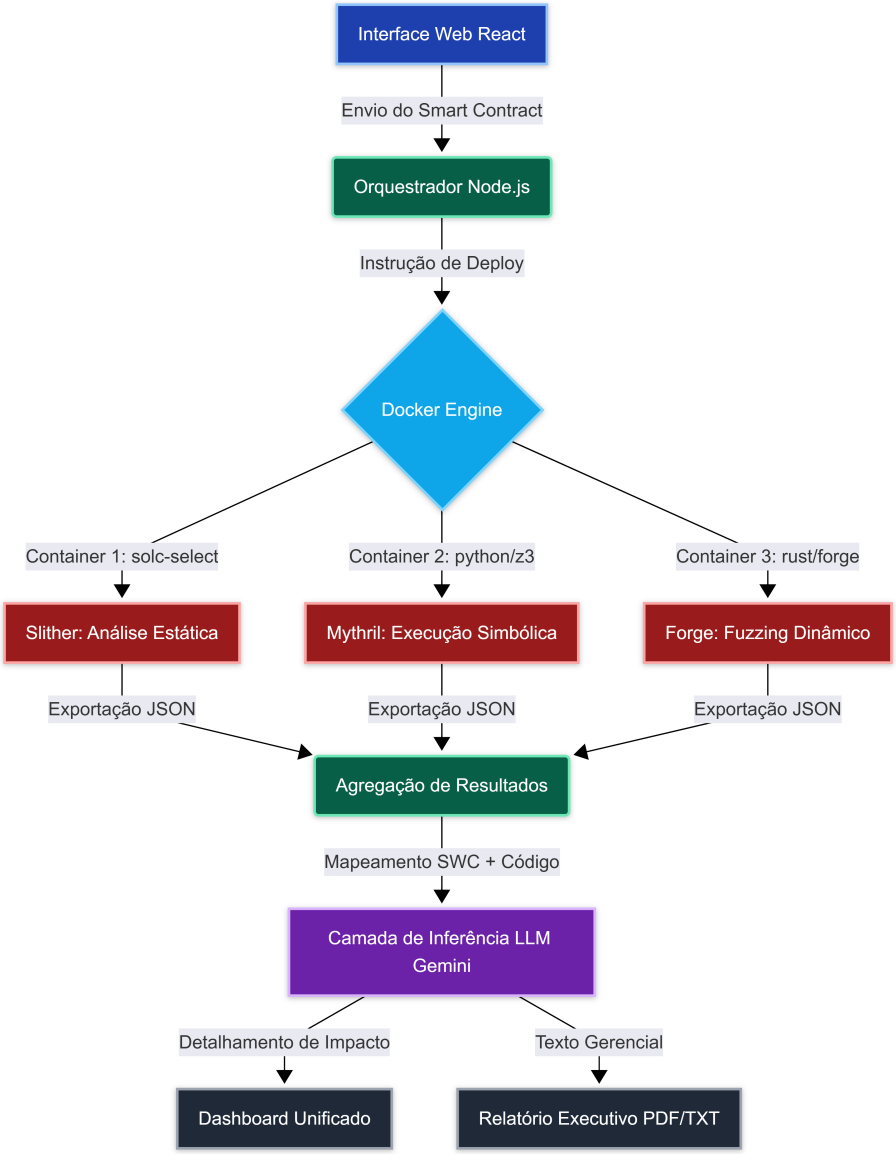

Figura 1. Fluxograma da arquitetura orquestrada do framework. Os motores de análise operam em containers isolados de forma assíncrona.

Relatório de Auditoria

RELATÓRIO DE AUDITORIA DE SEGURANÇA: CONTRATO "VULNERAVEL"

1. RESUMO EXECUTIVO

A auditoria identificou uma vulnerabilidade de Risco Crítico (Reentrância) e duas observações de caráter informativo relacionadas à infraestrutura do código. O problema principal permite que um atacante retire fundos do contrato repetidamente, potencialmente esvaziando todo o saldo depositado por outros usuários.

A segurança atual do contrato é considerada insuficiente para operação em ambiente de produção (Mainnet) até que as correções sejam aplicadas.

---

2. IMPACTO DE NEGÓCIO

A exploração das falhas detectadas pode resultar em:

- Perda Total de Ativos: Um invasor pode drenar 100% dos fundos custodiados no contrato.
- Dano à Reputação: A perda de fundos de usuários em contratos inteligentes é pública e irreversível.
- Inviabilidade Operacional: Erros no compilador podem causar comportamentos imprevistos.

---

3. EXEMPLOS DE CENÁRIOS

Cenário A: O Ataque de Reentrância (o "Saque Infinito")

1. Usuário solicita saque
2. Contrato envia dinheiro
3. Antes de atualizar saldo, ocorre nova chamada
4. Repetição até esvaziar o contrato

Cenário B: Instabilidade do Compilador

Uso da versão Ô.8.0 pode conter bugs conhecidos.

---

4. RECOMENDAÇÕES

1. Aplicar padrão Checks-Effects-Interactions
2. Usar ReentrancyGuard
3. Fixar versão do compilador (ex: 0.8.19)

Figura 2. Relatório executivo gerado com o auxílio de Inteligência Artificial (LLM), traduzindo as falhas técnicas em impacto de negócio e sugestões de mitigação.

5. Resultados

Os resultados preliminares evidenciam que a orquestração unificada do ecossistema de auditoria supera significativamente as abordagens de detecção isoladas, validando a complementaridade entre a análise estática e a execução simbólica. Somado a isso, a geração automatizada de relatórios com o suporte de LLMs otimiza substancialmente a tomada de decisão do auditor. Por fim, a arquitetura garante que o profissional possua o ferramental necessário para validar os apontamentos das ferramentas, utilizando o Foundry para a execução de testes dinâmicos.

5.1. Dataset e Ambiente de Avaliação

Para a condução dos testes, foi estruturado um dataset com 53 contratos inteligentes. A base primária para a extração das amostras foi o repositório SmartBugs, um banco de dados já consolidado na literatura para avaliar ferramentas de auditoria. O conjunto selecionado cobre as principais falhas mapeadas pelo Smart Contract Weakness Classification (SWC).

A escolha por um conjunto menor, de 53 amostras permitiu validar manualmente cada vulnerabilidade e analisar a fundo o comportamento dos testes dinâmicos no Foundry e as respostas da LLM. Como a execução de testes dinâmicos e o processamento via inteligência artificial demandam alto custo computacional, essa delimitação foi necessária para garantir uma análise qualitativa detalhada dos resultados.

5.2. Métricas de Avaliação

As métricas de desempenho avaliadas para o *benchmark* da ferramenta incluem:

- **Precisão (P) e *Recall* (R):** Medição da taxa de Falsos Positivos (FP), Falsos Negativos (FN) e Verdadeiros Positivos (VP) frente às vulnerabilidades do *dataset*. O objetivo central é demonstrar que a validação cruzada entre análise estática (Slither), execução simbólica (Mythril) e triagem contextual (LLM) reduz substancialmente o ruído global gerado pela execução isolada.
- **Pontuação F_1 (*F1-Score*):** Cálculo da média harmônica entre a precisão e o *recall* para avaliar o equilíbrio e a confiabilidade global do sistema de detecção proposto.
- **Latência de Auditoria e Triagem:** Comparação qualitativa entre o tempo gasto para processar, validar e gerar um relatório unificado via plataforma *versus* a execução descentralizada e a triagem manual de *logs* brutos de cada *Interface de Linha de Comando* (CLI).

As métricas de Precisão (P) e *Recall* (R) são fundamentais para o cálculo da Pontuação F_1, que valida a eficácia do artefato, conforme as equações matemáticas padrão da literatura:

$$P = \frac{VP}{VP + FP} \tag{1}$$

$$R = \frac{VP}{VP + FN} \tag{2}$$

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{3}$$

A Tabela 1 apresenta uma comparação quantitativa do desempenho das abordagens de auditoria analisadas análise estática (Slither), execução simbólica (Mythril) e o framework proposto (OrchestralSec) considerando um conjunto de 53 contratos inteligentes. As métricas avaliadas incluem Verdadeiros Positivos (VP), Falsos Positivos (FP), Falsos Negativos (FN), além de Precisão, Recall e F1-Score. Observa-se que o framework OrchestralSec obteve o melhor desempenho geral, alcançando os maiores valores de recall (93,88%) e F1-Score (92,93%), o que indica maior capacidade de identificar vulnerabilidades com equilíbrio entre precisão e cobertura. Em contraste, a análise estática apresentou menor recall (60,00%), evidenciando limitações na detecção de determinadas falhas, enquanto a execução simbólica demonstrou desempenho intermediário, com bons resultados tanto em precisão quanto em recall.

Tabela 1. Desempenho Comparativo das Abordagens de Auditoria (53 Contratos)

Abordagem	**VP**	**FP**	**FN**	**Precisão (%)**	**Recall (%)**	**F1-Score (%)**
Análise Estática (Slither)	30	3	20	90.90	60.00	72.28
Execução Simbólica (Mythril)	42	5	6	89.36	87.50	88.42
Framework (OrchestralSec)	46	4	3	92.00	93.88	92.93

Tabela 2. Taxa de Detecção (Recall) por Categoria de Vulnerabilidade

Categoria de Vulnerabilidade	**Amostras**	**Slither**	**Mythril**	OrchestralSec
Negação de Serviço (*DoS*)	6	100% (6/6)	100% (6/6)	100% (6/6)
Reentrância	5	80.0% (4/5)	80.0% (4/5)	100% (5/5)
Aleatoriedade Insegura	5	80.0% (4/5)	60.0% (3/5)	100% (5/5)
Short Addresses	1	0.0% (0/1)	100% (1/1)	100% (1/1)
Chamadas de Baixo Nível	10	60.0% (6/10)	90.0% (9/10)	90.0% (9/10)
Controle de Acesso	9	55.5% (5/9)	88.8% (8/9)	88.8% (8/9)
Erros Aritméticos (*Overflow*)	8	25.0% (2/8)	62.5% (5/8)	75.0% (6/8)
Front-Running	4	25.0% (1/4)	75.0% (3/4)	75.0% (3/4)
Manipulação de Tempo	5	40.0% (2/5)	60.0% (3/5)	60.0% (3/5)

Nota: Os percentuais representam a taxa de Verdadeiros Positivos (VPs) identificados em relação ao total de contratos vulneráveis da respectiva categoria.

A Tabela 2 detalha a taxa de detecção (recall) das abordagens avaliadas por categoria de vulnerabilidade, permitindo uma análise mais granular do desempenho de cada técnica. Observa-se que o framework OrchestralSec apresenta desempenho superior ou equivalente às demais abordagens na maioria das categorias, destacando-se especialmente na detecção de vulnerabilidades como reentrância, aleatoriedade insegura e erros aritméticos. Além disso, todas as abordagens obtiveram desempenho máximo na identificação de vulnerabilidades de negação de serviço (DoS). Por outro lado, a

análise estática (Slither) apresentou limitações significativas em categorias mais complexas, como erros aritméticos e front-running. Esses resultados reforçam a hipótese de que a combinação de múltiplas técnicas no framework proposto contribui para uma cobertura mais ampla e eficaz na identificação de vulnerabilidades em contratos inteligentes.

6. Conclusão e Trabalhos Futuros

Este trabalho apresentou o desenvolvimento e a avaliação de um Framework Integrado para auditoria de contratos inteligentes, fundamentado na orquestração de técnicas de análise estática, execução simbólica e testes dinâmicos assistidos por Inteligência Artificial. A hipótese central de que a abordagem híbrida seria capaz de mitigar as limitações inerentes às ferramentas individuais foi confirmada por meio de validação empírica. A análise conduzida sobre um *dataset* derivado do repositório SmartBugs, composto por 53 amostras entre vulnerabilidades reais e sintéticas, evidenciou ganhos expressivos de desempenho, com o *framework* alcançando uma pontuação F_1 de 92,93%, superior aos resultados obtidos isoladamente por ferramentas como Slither (72,28%) e Mythril (88,42%).

Além dos avanços quantitativos, a proposta contribui qualitativamente ao promover a unificação de *outputs* heterogêneos por meio do padrão SARIF e ao empregar Modelos de Linguagem de Grande Escala (LLMs) na transformação de *logs* técnicos em relatórios contextuais e acionáveis. A integração com testes de *fuzzing*, via Foundry, reforça a capacidade de validação prática das vulnerabilidades identificadas, reduzindo a carga cognitiva do auditor e permitindo uma atuação mais estratégica. Dessa forma, o *framework* demonstra-se tecnicamente viável, confiável e relevante para o aumento da segurança no desenvolvimento de contratos inteligentes em *Solidity*.

Como perspectivas futuras, destaca-se a ampliação das capacidades do sistema com a geração automatizada de provas de conceito por meio de LLMs, a implementação de mecanismos de remediação autônoma com validação por testes de regressão, a integração em pipelines de CI/CD para suporte a práticas de DevSecOps e a expansão do ecossistema de ferramentas, incluindo motores de *fuzzing* guiado por propriedades e técnicas de verificação formal, visando ampliar a cobertura e a robustez das análises em cenários de maior complexidade.

Agradecimentos

Os autores agradecem o apoio concedido pelo Ministério da Ciência, Tecnologia e Inovação (MCTI) por meio de duas fontes: 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), e recursos financeiros do FUNTTEL, administrados pela FINEP, especificamente no âmbito do projeto ”Saúde 5G - Segurança, Privacidade, Inclusão e Qualidade em Telemedicina no Contexto da Web 3.0”(Convênio nº 01.23.0468.00, Referência 0844/23).

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.

Atzei, N., Bartoletti, M., and Cimoli, T. (2017). A survey of attacks on ethereum smart contracts (sok). In *Principles of Security and Trust: 6th International Conference, POST 2017*, pages 164–186. Springer.

Braga, A. M., Marino, F. C. H., and dos Santos, R. R. (2017). Segurança de aplicações blockchain além das criptomoedas. In *Anais do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2017)*, chapter 3. SBC, Brasília. Minicursos.

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.

ConsenSys Diligence (2024). Mythril: Security analysis tool for evm bytecode. `https://github.com/ConsenSysDiligence/mythril`. Acessado em: 04 fev. 2026.

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.

Fonseca, F. M., Silva, P. H. G., Mendonça, D., and Moura, M. d. S. (2025). Towards the evolution of tools for detecting vulnerabilities in smart contracts: A case study of mythril and slither. In *Anais do Workshop em Blockchain: Teoria, Tecnologias e Aplicações (WBlockchain)*. SBC.

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.