Análise das Ferramentas de Detecção de Vulnerabilidades para Contratos Inteligentes de Blockchains EVM

Josué N. Campos[1], Luís H. S. de Carvalho[1], Isdael R. Oliveira[2], Aline C. S. Silva[1], Iago G. Falcão[1], Matheus J. da Silva[1], Glauber D. Gonçalves[2], Alex B. Vieira[3], José A. M. Nacif[1]

[1] Universidade Federal de Viçosa (UFV) – Florestal, MG – Brasil
[2]Universidade Federal do Piauí (UFPI) – Teresina, PI – Brasil
[3]Universidade Federal do Juiz de Fora (UFJF) – Juiz de Fora, MG – Brasil

{josue.campos, luis.h.carvalho, aline.c.silva}@ufv.br
{iago.falcao, matheus.junio, jnacif}@ufv.br
{isdael, ggoncalves}@ufpi.edu.br
alex.borges@ufjf.edu.br

Abstract. *Smart contracts have introduced security challenges due to their complexity in creating programmable financial transactions and the immutable nature of blockchain networks. During the last few years, several automated vulnerability detection tools targeted mitigating these risks. However, the effectiveness of these tools varies significantly, and detection capabilities remain a critical area of research. This paper presents a comparative study of the leading open-source tools from previous works. Using a curated benchmark, we evaluate the performance of these tools against the OWASP security vulnerabilities for smart contracts in 2025. Our results show that the tools have high variability in their vulnerability detection capabilities, opening opportunities for future research.*

Resumo. *Os contratos inteligentes introduziram desafios de segurança devido à sua liberdade para criar operações financeiras programáveis e devido à natureza imutável das redes blockchain. Para mitigar estes riscos, várias ferramentas de verificação automática de vulnerabilidades foram desenvolvidas. Contudo, a efetividade destas ferramentas varia significativamente e a capacidade de detecção de vulnerabilidades em contratos inteligentes continua sendo uma área crítica de pesquisa. Neste trabalho, nós apresentamos um estudo comparativo das principais ferramentas de código-aberto da literatura. Por meio de um benchmark previamente auditado, avaliamos o desempenho destas ferramentas em relação às vulnerabilidades OWASP para contratos inteligentes em 2025. Nossos resultados mostram que as ferramentas possuem alta variabilidade em detectar as vulnerabilidades, abrindo oportunidades para pesquisas futuras.*

1. Introdução

O aumento da demanda por soluções financeiras descentralizadas envolvendo as redes blockchain levou ao surgimento dos contratos inteligentes, programas escritos em uma linguagem Turing completa que executam operações de gerenciamento de ativos e acordos na blockchain automaticamente [Buterin et al. 2013]. A linguagem de programação

Solidity, a linguagem mais adotada para o desenvolvimento de contratos inteligentes, introduziu desafios de segurança únicos devido à sua liberdade para criar operações financeiras programáveis e devido à natureza imutável dos contratos inteligentes uma vez que eles são implantados em redes blockchains [Wohrer and Zdun 2018]. Vulnerabilidades que podem ser exploradas por agentes maliciosos como reentrada e questões de controle de acesso de permissões resultaram em perdas financeiras significativas no ecossistema descentralizado proposto pelas blockchains [Mense and Flatscher 2018].

Para mitigar estes riscos, várias ferramentas automáticas de verificação de vulnerabilidades foram desenvolvidas para analisar contratos inteligentes escritos na linguagem Solidity [Khan and Namin 2024]. Estas ferramentas utilizam técnicas como análise estática e dinâmica para identificar vulnerabilidades de segurança antes da implantação destes contratos na rede. Contudo, a efetividade destas ferramentas varia significativamente e a capacidade de detecção continua sendo uma área crítica de pesquisa [Ressi et al. 2024]. Os trabalhos da literatura que relacionam estas ferramentas desconsideram que muitas acabam sendo descontinuadas ou passam a ser incorporadas em empresas de auditoria e deixam de receber suporte em código-aberto [Kushwaha et al. 2022a].

Neste artigo, nós avaliamos a capacidade de detecção de vulnerabilidades das principais ferramentas de código-aberto que analisam contratos inteligentes em Solidity. Nós empiricamente avaliamos o desempenho destas ferramentas por meio de um *benchmark* da literatura [Di Angelo and Salzer 2023] de contratos inteligentes vulneráveis previamente auditado manualmente. Este estudo visa destacar o estado-da-arte da análise automática de segurança de contratos inteligentes, identificar áreas de melhorias, potencializar o desenvolvimento de pesquisas futuras e novas ferramentas.

Nós resumimos as nossas principais contribuições a seguir:

- Constatamos que as ferramentas possuem uma alta variabilidade em identificar as vulnerabilidades de segurança, atingindo uma faixa de 0 a 92%;
- Observamos que os conjuntos de dados de contratos inteligentes vulneráveis da literatura possuem versões antigas da linguagem Solidity em sua maioria, com 70% dos contratos descontinuados pelas ferramentas;
- Direcionamos pesquisas futuras a partir da análise da acurácia das ferramentas e constatando que novas técnicas e ferramentas precisam ser desenvolvidas para cobrir todas as vulnerabilidades com confiabilidade, robustez e eficiência.

O restante deste artigo está organizado da seguinte maneira: a Seção 2 discute os principais conceitos associados ao trabalho. A Seção 3 apresenta os trabalhos da literatura que se relacionam com o nosso. Já a Seção 4, discute a metodologia adotada para obter os resultados da Seção 5. Por fim, a Seção 6 apresenta nossas considerações finais e direções futuras.

2. Visão Geral

Esta seção discute os principais conceitos associados ao trabalho, *i.e.*, contratos inteligentes, vulnerabilidades de segurança e as ferramentas de detecção automática de vulnerabilidades.

2.1. Contratos Inteligentes

Um contrato inteligente é um programa desenvolvido em uma linguagem de alto nível. Em redes blockchains EVM (*Ethereum Virtual Machine*), os contratos são escritos em Solidity. Uma blockchain EVM pode ser definida como uma rede que suporta a máquina virtual da Ethereum, camada responsável por executar o código de um contrato inteligente. Este código é compilado em *bytecodes* e implantado na blockchain por meio de uma transação. Nesta transação, o dado enviado corresponde ao *bytecode* do contrato [Wood 2014]. Após a conclusão da transação, o contrato recebe um endereço único na blockchain, assim como uma conta de usuário, ou seja, a partir disso qualquer interação com o contrato inteligente modifica o estado da rede blockchain. Qualquer usuário pode interagir com o contrato enviando uma transação para este endereço gerado [Harvey et al. 2021].

Os contratos inteligentes executam automaticamente acordos entre partes, eliminando a necessidade de intermediários [Campos et al. 2024]. Além de serem públicos e imutáveis (o que confere transparência e segurança nas transações), os contratos possuem regras e condições estabelecidas pela blockchain utilizada para controlar as interações entre as partes envolvidas [Egelund-Müller et al. 2017].

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.20;
3
4  contract ExemploCarteira {
5      address public dono;
6      constructor() {dono = msg.sender;}
7      receive() external payable {}
8      function saque(uint256 quantidade) external {
9          require(msg.sender == dono, "Somente o dono pode
               sacar");
10         require(address(this).balance >= quantidade, "
               Saldo insuficiente");
11         payable(dono).transfer(quantidade);
12     }
13 }
```

Algoritmo 1. Exemplo de código-fonte de um contrato inteligente em Solidity.

O Algoritmo 1 é um exemplo de contrato inteligente escrito na linguagem Solidity utilizada na rede Ethereum. Neste exemplo, o código do contrato inteligente é responsável por simular uma carteira capaz de receber e sacar valores em *Ether* (ETH), criptomoeda nativa da rede Ethereum. A função *receive* nativa da linguagem permite o contrato receber ETH de qualquer endereço e, por sua vez, a função "saque" permite o dono da carteira retirar os valores depositados no contrato. Soluções como esta e entre outras geraram oportunidades para a existência de inúmeras aplicações descentralizadas, *e.g.*, *tokens* e corretoras de ativos financeiros [Mendonça et al. 2022].

2.2. Vulnerabilidades de Segurança

Em contratos inteligentes, vulnerabilidades surgem quando há uma falha no código que pode ser explorada de maneira adversa. Tais falhas podem ocorrer devido a erros de

programação, má definição de regras de execução ou interações inseguras com outros contratos [Nethermind 2025]. Diferente de sistemas tradicionais, onde vulnerabilidades podem ser corrigidas por meio de atualizações de software, contratos inteligentes implantados na blockchain são imutáveis, o que significa que qualquer erro presente no momento da implantação pode persistir por tempo indeterminado e ser explorado por agentes maliciosos [Kushwaha et al. 2022b].

Nesse sentido, a padronização de codificação de contratos seguros, bem como a relação das vulnerabilidades existentes relacionadas à blockchain e contratos inteligentes continua um desafio em aberto [Ressi et al. 2024]. Uma solução não descontinuada da literatura é a OWASP (*Open Web Application Security Project*) [OWASP 2025a], que desempenha o papel de fornecer diretrizes, práticas recomendadas e classificações de vulnerabilidades específicas da área de blockchain. O ranque *OWASP Smart Contract Security Top 10* [OWASP 2025b] lista as principais vulnerabilidades encontradas em contratos inteligentes. Essa classificação auxilia desenvolvedores, auditores e pesquisadores a compreenderem os riscos mais comuns e a implementarem medidas preventivas.

Tabela 1. Top-10 vulnerabilidades de contratos inteligentes em 2025 (Adaptado de [OWASP 2025b]).

Código	Vulnerabilidade	Descrição
SC01	Controle de Acesso	Permite que usuários não autorizados modifiquem dados ou funções devido à falta de verificações de permissão.
SC02	Manipulação de Preço de Oráculo	Exploração de oráculos de preço para alterar a lógica do contrato, resultando em perdas financeiras.
SC03	Erros de Lógica	Erros na lógica do contrato que levam a comportamento inesperado, como distribuição incorreta de recompensas.
SC04	Falta de Validação de Entrada	Falta de validação de entrada permite que atacantes manipulem a execução do contrato.
SC05	Ataques de Reentrada	Permite múltiplas execuções de uma função antes de sua conclusão, podendo drenar fundos do contrato.
SC06	Chamadas Externas Não Verificadas	Falha ao verificar chamadas externas pode levar a execução incorreta do contrato.
SC07	Ataques de Empréstimos Rápidos	Uso de empréstimos rápidos para manipular protocolos, drenando liquidez ou alterando preços.
SC08	Overflow e Underflow de Inteiros	Erros aritméticos que podem levar a cálculos incorretos ou roubo de *tokens*.
SC09	Aleatoriedade Insegura	Falta de aleatoriedade segura pode permitir previsibilidade em sorteios e distribuições de *tokens*.
SC10	Ataques de Negação de Serviço (DoS)	Exploração de consumo excessivo de recursos para tornar o contrato inoperante.

A Tabela 1 fornece um panorama das vulnerabilidades mais críticas em contratos inteligentes para o ano de 2025. Essas falhas podem comprometer a segurança de aplicações descentralizadas, resultando em perdas financeiras e instabilidades em redes blockchains. Vulnerabilidades como SC01, SC03, SC04, SC05, SC06, SC08 e SC10 são falhas que estão intimamente ligadas com a maneira de desenvolver as funcionalidades de um contrato inteligente. Por outro lado, vulnerabilidades como SC02, SC07 e SC09 são falhas que podem afetar a execução de um contrato inteligente, porém a detecção pode requerer o monitoramento da própria blockchain em que o contrato está implantado,

resultando em ataques mais difíceis de serem detectados e mitigados.

2.3. Ferramentas de Detecção de Vulnerabilidades

Em [Khan and Namin 2024] os autores destacam que as ferramentas automatizadas de detecção de vulnerabilidades são essenciais para impedir ataques e perdas financeiras em contratos inteligentes. Estas ferramentas são apresentadas como uma resposta ao crescimento de fraudes e ataques que ameaçam o desenvolvimento e a adoção dos contratos inteligentes no ecossistema de redes blockchains.

As principais técnicas de análise podem ser agrupadas em análise estática e dinâmica [Praitheeshan et al. 2019]. A análise estática verifica o código sem executá-lo, e abrange métodos como a execução simbólica. Neste tipo de técnica, o *bytecode* é processado de forma abstrata com substituição de variáveis por símbolos, interpretação sistemática, atualização do estado de execução e verificação das restrições de caminho via solucionadores para identificar padrões de vulnerabilidade [Qian et al. 2022, Rameder et al. 2022]. A Slither é uma das principais ferramentas que emprega este tipo de análise ao receber o contrato inteligente como entrada e detectar vulnerabilidades relacionadas às funções implementadas [Feist et al. 2019].

Por outro lado, a análise dinâmica avalia o comportamento de um contrato inteligente em tempo real, empregando técnicas como o *fuzzing*, que submete o código à condições de entrada diversificadas para revelar vulnerabilidades [Rameder et al. 2022]. Neste tipo de análise, há a possibilidade de identificação de vulnerabilidades que podem não ser detectadas na análise estática. O processo envolve geração sistemática de casos de teste, execução, monitoramento de estados e análise de exceções [Li et al. 2023]. Para este tipo de técnica, a ferramenta ConFuzzius mostra-se como uma das principais ao integrar a técnica de *fuzzing* com algoritmos genéticos que melhoram a cobertura de código [Torres et al. 2021].

Além das técnicas convencionais de análise estática e dinâmica, técnicas como a verificação formal e aprendizado de máquina também são empregadas no contexto de contratos inteligentes [Ben Fekih et al. 2025]. Métodos de verificação formal baseiam-se em resolver equações matemáticas e lógicas baseadas no código do contrato para determinar a ausência ou presença de uma vulnerabilidade, sendo uma técnica eficiente em eliminar falsos positivos apesar da difícil implementação [Almakhour et al. 2020]. Ferramentas como a Mythril emprega a técnica de verificação formal em conjunto com a execução simbólica para avaliar caminhos de execuções vulneráveis [ConsenSys 2018]. Já as abordagens baseadas em aprendizado de máquina, especialmente o aprendizado profundo, emergem como métodos promissores para detectar vulnerabilidades que possuem padrões conhecidos, embora enfrentem desafios quanto à disponibilidade de conjunto de dados adequados [Qian et al. 2022, Rameder et al. 2022].

3. Trabalhos Relacionados

Nesta seção, nós apresentamos os trabalhos relacionados que abordam a detecção de vulnerabilidades em contratos inteligentes e avaliam ferramentas de análise automática. Nós agrupamos os trabalhos em duas categorias: Revisão de ferramentas e conjunto de dados e Análises de desempenho.

Revisão de ferramentas e conjuntos de dados. Os trabalhos existentes que avaliam as ferramentas de detecção de vulnerabilidades abordam uma relação entre as técnicas que cada ferramenta utiliza e quais vulnerabilidades cada uma é capaz de detectar [Rameder et al. 2022, Zhou et al. 2022, Khan and Namin 2024]. Por meio da análise sistemática de trabalhos da literatura, os autores destes trabalhos destacam as principais técnicas que as ferramentas de verificação automática utilizam para identificar vulnerabilidades em contratos inteligentes. Apesar dos autores destacarem diferentes ferramentas e vulnerabilidades, o nosso trabalho estende essas análises comparando as principais ferramentas de código-aberto ainda em manutenção em relação às principais ameaças de 2025.

Análises de desempenho. Comparar o desempenho entre ferramentas que utilizam diferentes técnicas de verificação de vulnerabilidades é um desafio já proposto em trabalhos como [Parizi et al. 2018, Durieux et al. 2020, Kushwaha et al. 2022a]. Os autores destes trabalhos abordam a capacidade de detecção das ferramentas em relação às vulnerabilidades de segurança em contratos inteligentes. Nestes trabalhos são apresentadas as acurácias de cada ferramenta para cada vulnerabilidade e, apesar de em nosso trabalho utilizarmos desta metodologia, nós estendemos essa análise utilizando um conjunto de dados de contratos inteligentes previamente auditado manualmente, bem como analisamos somente as ferramentas que ainda são atualizadas em seus repositórios.

Tabela 2. Contribuições em relação aos trabalhos relacionados.

Trabalho	*Dataset* auditado manualmente	Ferramentas atualizadas	Top-10 OWASP
[Parizi et al. 2018]	●	◐	◐
[Durieux et al. 2020]	○	◐	◐
[Kushwaha et al. 2022a]	○	◐	◐
[Rameder et al. 2022]	◐	◐	◐
[Zhou et al. 2022]	○	○	◐
[Khan and Namin 2024]	○	●	◐
Este trabalho	●	●	●

Conforme a Tabela 2, nosso trabalho diferencia dos demais acima apresentando uma análise das ferramentas ainda atualizadas de acordo com as vulnerabilidades atuais de contratos inteligentes. Além disso, apresentamos uma análise comparativa em relação à um conjunto de dados consolidado de contratos inteligentes auditados manualmente. Dessa maneira, é possível relacionar as vantagens e desvantagens de cada ferramenta, assim como o estado-da-arte da verificação automática de vulnerabilidades em contratos inteligentes de blockchains EVM.

4. Metodologia

Nesta seção apresentamos as metodologias aplicadas para compararmos as principais ferramentas de detecção automática de vulnerabilidades, assim como o conjunto de dados utilizado para validação do nosso estudo.

4.1. Conjunto de Dados

Alguns conjuntos de dados existentes na literatura foram gerados a partir de diagnósticos automatizados de ferramentas de análise, que, embora úteis, podem conter falsos positivos e falsos negativos, comprometendo a confiabilidade das medições [Ressi et al. 2024]. Este cenário torna desafiadora a avaliação rigorosa de modelos de detecção de vulnerabilidades, uma vez que a ausência de rótulos verificados por especialistas impacta diretamente na qualidade dos *benchmarks* utilizados.

Entretanto, para este trabalho, nós utilizamos o conjunto de dados construído em [Di Angelo and Salzer 2023]. Os autores descrevem um processo detalhado de unificação e consolidação dos principais conjuntos de dados da literatura. Em primeiro lugar, os autores coletaram diversos *benchmarks* contendo contratos inteligentes e suas respectivas vulnerabilidades e identificaram informações como endereços, código-fonte, *bytecode* e classificações de segurança. Em seguida, aplicaram estratégias para mapear entradas duplicadas, eliminar falsos positivos e negativos e padronizar as vulnerabilidades. O resultado é um conjunto de dados mais estruturado, abrangente e menos redundante, que pode ser utilizado para avaliar ferramentas de segurança de forma mais rigorosa.

Nós selecionamos apenas as vulnerabilidades presentes no ranque OWASP da Tabela 1. Esta escolha foi feita com o objetivo de avaliarmos a capacidade das ferramentas em identificar as falhas mais críticas e atuais na indústria de segurança em blockchain. Ao restringir o escopo às vulnerabilidades documentadas pela OWASP, nós garantimos uma avaliação alinhada com as melhores práticas de segurança existentes. A partir deste filtro, o conjunto de dados utilizado conta com um total de 6.072 contratos verificados manualmente. A Figura 1 exibe a quantidade de contratos inteligentes para cada tipo de vulnerabilidade. Observa-se que SC06, SC08 e SC10 são as vulnerabilidades mais frequentes, compondo mais de 50% do conjunto. Já a vulnerabilidade SC04 possui a menor ocorrência, sendo referenciada em apenas 60 contratos. De qualquer forma, esses dados auxiliam na identificação de quais vulnerabilidades são mais comuns nos contratos analisados.

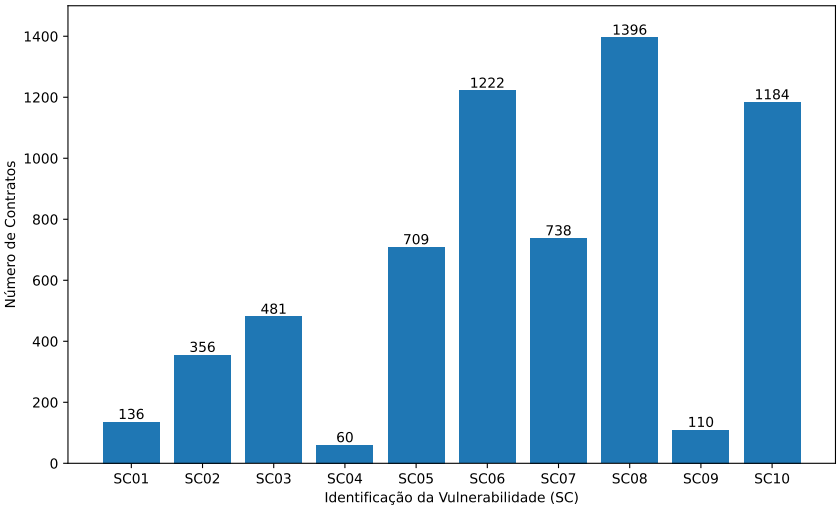

Figura 1. Quantidade de contratos inteligentes por vulnerabilidade.

4.2. Ferramentas Selecionadas

Para a condução deste estudo comparativo das ferramentas de detecção de vulnerabilidades em contratos inteligentes na blockchain Ethereum, nós selecionamos três ferramentas representativas que empregam diferentes abordagens de análise: Slither, Mythril, ConFuzzius.

Cada ferramenta foi selecionada por suas características particulares e abordagens distintas para análise de segurança. O Slither representa as ferramentas de análise estática baseadas em código-fonte [Feist et al. 2019]. Já o Mythril exemplifica técnicas de execução simbólica ao nível de *bytecode* [ConsenSys 2018], e o ConFuzzius ilustra a abordagem híbrida com *fuzzing* evolutivo [Torres et al. 2021]. Esta diversidade metodológica permite uma comparação representativa do estado atual das principais técnicas de análise de segurança para contratos inteligentes. Além disso, apesar de em trabalhos anteriores como [Kushwaha et al. 2022a] serem apresentadas mais ferramentas disponíveis na literatura, nós constatamos que ferramentas como Oyente [Luu et al. 2016], Vandal [Brent et al. 2018], ContractFuzzer [Jiang et al. 2018], foram descontinuadas e não recebem mais atualizações. Dessa maneira, comparamos as principais ferramentas que ainda recebem atualizações e lidam com as versões mais atuais da linguagem Solidity.

A ferramenta Slither converte o código Solidity em uma representação intermediária chamada SlithIR, que adota a forma *Static Single Assignment* (SSA) e um conjunto reduzido de instruções, a fim de facilitar a implementação de análises mantendo a semântica do código [Feist et al. 2019]. A ferramenta emprega técnicas de análise de fluxo de dados e rastreamento de informações sensíveis (*taint tracking*) para detectar vulnerabilidades que dizem respeito às funcionalidades implementadas de um contrato inteligente.

Já a ferramenta Mythril realiza análises através de três abordagens principais: execução simbólica, resolução SMT (*Satisfiability Modulo Theories*) e análise de propagação de dados sensíveis (*taint analysis*) [Kushwaha et al. 2022a]. O módulo de execução simbólica LASER é responsável por simular o ambiente de execução dos contratos, permitindo que Mythril explore todos os estados possíveis de execução do contrato ao longo de múltiplas transações. A ferramenta utiliza também o teorema provador Z3, desenvolvido pela Microsoft Research, para validar ou refutar a existência de estados comprometidos [Sharma and Sharma 2022].

O ConFuzzius apresenta uma estrutura composta por três componentes principais: o módulo de *fuzzing* evolutivo, a Máquina Virtual Ethereum (EVM) com instrumentação e o componente de análise de traços de execução [Torres et al. 2021]. O componente evolutivo da ferramenta utiliza um algoritmo genético para gerar sucessivas populações de casos de teste e intercala técnicas de *fuzzing* com execução simbólica quando encontra ramificações complexas no código.

5. Resultados

Neste estudo analisamos as ferramentas Slither, Mythril e ConFuzzius para a detecção de vulnerabilidades em contratos inteligentes. A análise considerou os contratos específicos para cada vulnerabilidade listada na OWASP, conforme a Tabela 1. Nossos resultados

indicam como cada ferramenta possui vantagens e desvantagens dependendo do caso de uso. A ConFuzzius apresenta-se como a ferramenta mais eficiente em tempo de execução, porém possui poucos (4) detectores de vulnerabilidades em relação às outras ferramentas. Por sua vez, a ferramenta Mythril apresenta detectores para todas (10) as vulnerabilidades estudadas, porém enfrenta desafios de escalabilidade para auditorias de vários contratos ao atingir tempos de execução altos em relação às outras ferramentas. Por fim, a ferramenta Slither apresenta tempos de execução próximos ao do ConFuzzius e cobre quase todas as vulnerabilidades estudadas (7), mostrando-se como uma ferramenta balanceada no quesito de escalabilidade *versus* capacidade de detecção.

5.1. Capacidade de Detecção

Em relação a capacidade de detecção das vulnerabilidades do Top-10 do OWASP, é possível observar a acurácia das ferramentas na Figura 2. De acordo com o gráfico, as ferramentas Slither e Mythril possuem a capacidade de detectar uma maior variedade de vulnerabilidades dentre as ferramentas estudadas. Contudo, em casos como as vulnerabilidade SC06, SC08 e SC10 a ferramenta ConFuzzius possui uma acurácia maior.

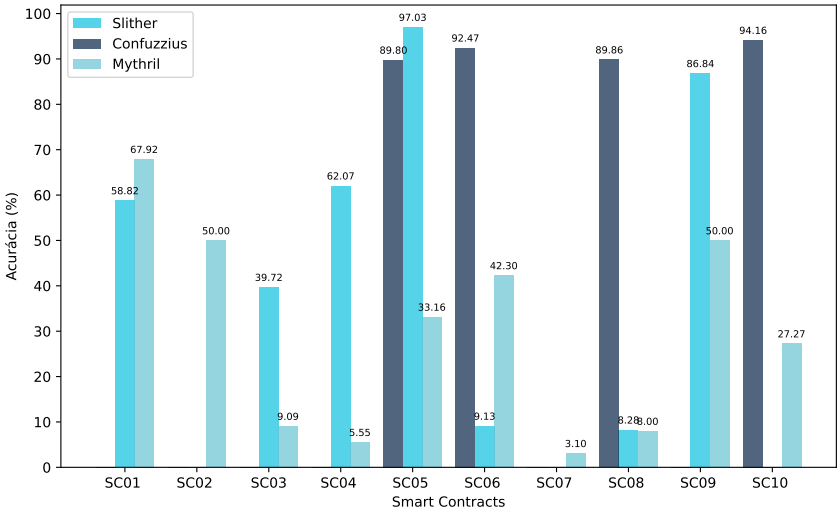

Figura 2. Comparação de Acurácia das ferramentas Slither, ConFuzzius e Mythril.

A ferramenta ConFuzzius não possui detectores para todas as vulnerabilidades do Top-10, porém possui a melhor acurácia de detecção para as vulnerabilidades que a ferramenta consegue suportar. É possível observar também que, a vulnerabilidade SC07 possui a menor capacidade de detecção dentre as ferramentas analisadas. Uma justificativa para este resultado é que esta vulnerabilidade em particular não depende somente do código do contrato, mas também da ordem de execuções das transações que ocorrem na blockchain.

Dentre as três ferramentas, a ferramenta Mythril é a que possui um desempenho menor, que é justificado na Seção 5.2. Para a maioria das vulnerabilidades a acurácia fica abaixo de 50%. Porém, ela foi a única ferramenta que conseguiu detectar a vulnerabilidade SC02, que é uma vulnerabilidade que também depende do fluxo de execução do contrato na blockchain. A capacidade da Mythril detectar vulnerabilidades deste tipo baseia-se em seu método de combinar a execução simbólica com a instrumentação de instruções EVM.

Já a Figura 3 apresenta o gráfico comparativo das três ferramentas em relação a quantidade de contratos analisados (CA), contratos verificados (CV), *i.e.* contratos inteligentes analisados que foram encontradas vulnerabilidades, contratos em que a execução teve erros (E) e, por fim os contratos não-verificados (CNV), que são os contratos analisados, mas que as vulnerabilidades deles não foram identificadas pelas ferramentas. É possível observar que a métrica de erros (E) representa mais da metade de todo o conjunto de dados.

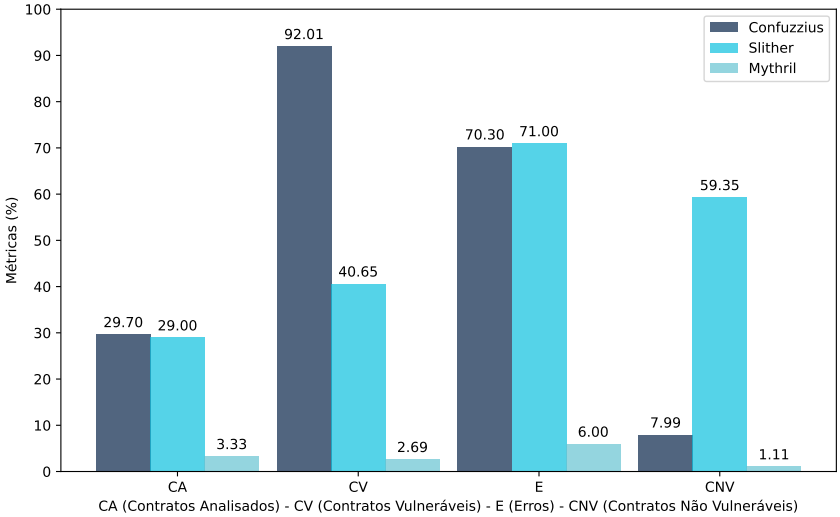

Figura 3. Análise de métricas das ferramentas Slither, ConFuzzius e Mythril.

Durante as nossas medições, observamos que muitos contratos do conjunto de dados possuem versões da linguagem Solidity muito antigas, e as ferramentas acabam não suportando, pois uma das principais boas práticas de codificação de contratos é utilizar versões atuais da linguagem Solidity. Versões atuais da linguagem podem mitigar certas vulnerabilidades pelo próprio compilador atualizado. Vale ressaltar que, como o conjunto de dados utilizado é uma consolidação de outros trabalhos da literatura, destacamos a criação de conjunto de dados de contratos inteligentes vulneráveis atuais como um desafio em aberto.

Entretanto, é possível observar pela métrica CV como o ConFuzzius apesar de detectar poucas vulnerabilidades do OWASP, possui a melhor taxa de detecção de contratos, mostrando a confiabilidade e robustez da ferramenta. Já o Slither, possui mais contratos não-verificados (CNV), porém mantém uma taxa melhor em CV em vista da Mythril, ou seja, a ferramenta Slither acaba sendo versátil em cobrir mais vulnerabilidades, fornecendo uma visão geral sobre o código de um contrato inteligente. A ferramenta Mythril não consegue executar todo o conjunto de dados em tempo hábil para gerar resultados, o que justifica as métricas serem inferiores para esta ferramenta.

5.2. Análise de Desempenho

A análise de desempenho das ferramentas ConFuzzius, Slither e Mythril revelou diferenças significativas tanto na cobertura de vulnerabilidades quanto na eficiência em tempo de execução, conforme a Tabela 3. A ferramenta ConFuzzius demonstrou uma abordagem mais limitada em relação às vulnerabilidades detectadas, cobrindo apenas

quatro das dez listadas no Top-10 da OWASP. No entanto, quando consegue identificar vulnerabilidades, apresenta uma acurácia alta, chegando a 89%, com um tempo médio de execução inferior a 2 segundos por contrato. Esse desempenho é reforçado pelos resultados obtidos ao executar a ferramenta no conjunto de dados analisado: dos 4.511 contratos testados, 29,7% foram analisados com sucesso e, desses, 92,01% continham vulnerabilidades detectadas pela ferramenta. Apenas 7,99% das vulnerabilidades passaram despercebidas. Entretanto, um grande percentual dos contratos (70,3%) não foi testado, devido à erros de versão da linguagem Solidity.

Por outro lado, a ferramenta Slither se mostrou mais abrangente, detectando sete das dez vulnerabilidades do Top-10 da OWASP. Apesar dessa vantagem, seu desempenho em termos de acurácia foi inferior. A acurácia ficou em 18,28%, e o tempo médio de execução foi inferior a 4 segundos por contrato, ou seja, mais lento que a ferramenta ConFuzzius. A Slither conseguiu analisar 4.114 contratos, dos quais 29% foram analisados com sucesso. No entanto, a taxa de identificação de vulnerabilidades foi bem menor: apenas 40,65% dos contratos testados apresentaram vulnerabilidades detectadas, enquanto 59,35% passaram despercebidas. Assim como no ConFuzzius, um grande número de contratos (71%) não foi testado por problemas similares de versão da linguagem.

Tabela 3. Tempo de execução das ferramentas por vulnerabilidades.

SC	ConFuzzius	Slither	Mythril
SC01	-	1,52s	34,49h
SC02	-	-	5,52h
SC03	-	3,07s	7,45h
SC04	-	4,09s	5,35h
SC05	1,89s	2,44s	40,48h
SC06	1,9s	1,96s	7,58h
SC07	-	-	3,39h
SC08	1,94s	2,09s	9,18h
SC09	-	2,93s	8,24h
SC10	1,99s	-	8,33h

Os resultados mostram que, enquanto Slither cobre mais tipos de vulnerabilidades, sua eficácia na detecção é menor. Por outro lado, ConFuzzius se sai melhor na identificação das vulnerabilidades que de fato cobre, com uma taxa de sucesso muito mais alta. No entanto, sua aplicação é mais limitada, pois consegue testar menos tipos de vulnerabilidades. Já a ferramenta Mythril, os resultados indicam que a ferramenta possui detectores para todas as dez vulnerabilidades listadas no Top 10 da OWASP. No entanto, seu desempenho variou significativamente entre as diferentes categorias de vulnerabilidades, com tempos de execução extremamente altos e baixa acurácia de detecção. A ferramenta apresentou uma cobertura média de apenas 12,79% em relação ao conjunto de dados utilizado, com tempos de execução que variaram de 3,39 horas (SC07) a 40,48 horas (SC05). Em algumas vulnerabilidades, como SC01 (38,97% de cobertura) e SC05 (27,64%), a Mythril demonstrou eficiência moderada, enquanto em outras, como SC10 (0,92%) e SC03 (2,28%), sua detecção foi limitada.

Os resultados indicam que, em média, apenas 12,79% dos contratos foram analisa-

dos com sucesso para a ferramenta Mythril. Dentre esses, a taxa de contratos vulneráveis detectados variou amplamente. Para os contratos que não foram testados, alguns dos fatores que interferiram nessas medições incluem: Erros devido às versões antigas da linguagem Solidity e o tempo alto para análise do contrato. Em uma análise de segurança, a Mythril demonstra que apesar de possuir uma variedade de detectores, não consegue finalizar a verificação de um contrato em tempo hábil, dificultando a usabilidade da ferramenta.

6. Considerações Finais

Este trabalho apresenta um estudo comparativo as ferramentas ConFuzzius, Slither e Mythril para a detecção automática de vulnerabilidades em contratos inteligentes. Nas análises realizadas, foi utilizado um conjunto de dados com contratos inteligentes auditados manualmente por pesquisadores e especialistas, considerando as vulnerabilidades mais comuns de acordo com a OWASP para o ano de 2025.

A ferramenta ConFuzzius teve um tempo médio inferior a 2 segundos por vulnerabilidade e, apesar de testar apenas uma fração pequena do conjunto de dados, obteve alta taxa de detecção (92,01%). Por outro lado, a ferramenta Slither testou uma quantidade maior de contratos, dos quais 29% foram analisados com sucesso, porém sua taxa de identificação de vulnerabilidades foi de apenas 40,65%, mantendo um tempo de execução semelhante. Já a ferramenta Mythril, embora ofereça detectores para todas as vulnerabilidades listadas, apresentou uma cobertura média de apenas 12,79% do conjunto de dados devido aos tempos de execução que aumentaram significativamente em comparação as outras duas (podendo chegar até 40,48 horas ao analisar determinada vulnerabilidade) e variações de cobertura que vão de 0,92% a 38,97%.

Para análises que exigem maior precisão dentro de um escopo limitado, o ConFuzzius se mostra mais confiável e robusto, enquanto o Slither pode ser utilizado para uma abordagem mais abrangente, apesar de sua menor eficácia. Para análises em larga escala, a ferramenta Mythril demonstra como sendo inutilizável pelo fato de não conseguir retornar resultados em tempos satisfatórios, porém possui versatilidade em identificar vulnerabilidades que saem do escopo do código do contrato. Desafios comuns, como erros de versão e incompatibilidades, apontam para a necessidade de conjunto de dados com contratos inteligentes vulneráveis mais atuais, assim como há a necessidade por ferramentas de código-aberto com diferentes técnicas e que sejam mantidas continuamente. De modo geral, os resultados evidenciam que cada ferramenta apresenta vantagens e limitações, exigindo pesquisas e a resolução destes desafios em aberto na área de análise automática de contratos inteligentes. Como direções futuras, elencamos a criação de novas ferramentas que combinam diferentes técnicas e vantagens das ferramentas analisadas, assim como a definição de conjuntos de dados que representam o contexto atual da linguagem de codificação de contratos inteligentes.

7. Agradecimentos

O presente trabalho foi realizado com o apoio do projeto ILIADA - A nova Internet da confiança, financiado pelo MCTI com recursos oriundos da Lei das TICs -Lei nº 8.248, de 23 de outubro de 1991, no âmbito do PPI-SOFTEX, coordenado pela Softex e publicado PDI 03, DOU 01245.023862/2022-14. Gostaríamos também de agradecer o apoio financeiro da CAPES, Fapemig e CNPq.

Referências

Almakhour, M., Sliman, L., Samhat, A. E., and Mellouk, A. (2020). Verification of smart contracts: A survey. *Pervasive and Mobile Computing*, 67:101227.

Ben Fekih, R., Lahami, M., Bradai, S., and Jmaiel, M. (2025). Formal verification of erc-based smart contracts: A systematic literature review. *IEEE Access*, 13:11396–11422.

Brent, L., Jurisevic, A., Kong, M., Liu, E., Gauthier, F., Gramoli, V., Holz, R., and Scholz, B. (2018). Vandal: A scalable security analysis framework for smart contracts. *arXiv preprint arXiv:1809.03981*.

Buterin, V. et al. (2013). Ethereum white paper. *GitHub repository*, 1(22-23):5–7.

Campos, J. N., Mendonça, R. D., Fontinele, A., de Carvalho, L. H. S., Oliveira, I. R., Cardoso, Í. W. F., Coelho, R., Freitas, A. E. S., Gonçalves, G. D., Nacif, J. A. M., and Vieira, A. B. (2024). Finanças descentralizadas em redes blockchain: Perspectivas sobre pesquisa e inovação em aplicações, interoperabilidade e segurança. In *Jornada de Atualização em Informática 2024*, volume 44 of *Congresso da Sociedade Brasileira de Computação*, pages 7–56. SBC, Porto Alegre, 43nd edition.

ConsenSys (2018). Mythril. `https://github.com/ConsenSys/mythril`. Acessado em: 2025.

Di Angelo, M. and Salzer, G. (2023). Consolidation of ground truth sets for weakness detection in smart contracts. In *International Conference on Financial Cryptography and Data Security*, pages 439–455. Springer.

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 ACM/IEEE 42nd International conference on software engineering*, pages 530–541.

Egelund-Müller, B., Elsman, M., Henglein, F., and Ross, O. (2017). Automated execution of financial contracts on blockchains. *Business & Information Systems Engineering*, 59:457–467.

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.

Harvey, C. R., Ramachandran, A., and Santoro, J. (2021). *DeFi and the Future of Finance*. John Wiley & Sons.

Jiang, B., Liu, Y., and Chan, W. K. (2018). Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, pages 259–269.

Khan, Z. A. and Namin, A. S. (2024). A survey of vulnerability detection techniques by smart contract tools. *IEEE Access*.

Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., and Lee, H.-N. (2022a). Ethereum smart contract analysis tools: A systematic review. *Ieee Access*, 10:57037–57062.

Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., and Lee, H.-N. (2022b). Systematic review of security vulnerabilities in ethereum blockchain smart contract. *Ieee Access*, 10:6605–6621.

Li, H., Dang, R., Yao, Y., and Wang, H. (2023). A review of approaches for detecting vulnerabilities in smart contracts within web 3.0 applications. *Blockchains*, 1(1):3–18.

Luu, L., Chu, D.-H., Olickel, H., Saxena, P., and Hobor, A. (2016). Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254–269.

Mendonça, R. D., Campos, J. N., Vieira, L. F. M., Vieira, M. A. M., Vieira, A. B., and Nacif, J. A. M. (2022). Tokens não fungíveis (nfts): Conceitos, aplicações e desafios. In *Minicursos do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 52–94, Porto Alegre, RS, Brasil. SBC.

Mense, A. and Flatscher, M. (2018). Security vulnerabilities in ethereum smart contracts. In *Proceedings of the 20th international conference on information integration and web-based applications & services*, pages 375–380.

Nethermind (2025). Smart contract vulnerabilities and mitigation strategies. Accessed: 2025.

OWASP (2025a). Open web application security project (owasp). Accessed: 2025.

OWASP (2025b). Owasp smart contract top 10 project. Accessed: 2025.

Parizi, R. M., Dehghantanha, A., Choo, K.-K. R., and Singh, A. (2018). Empirical vulnerability analysis of automated smart contracts security testing on blockchains. *arXiv preprint arXiv:1809.02702*.

Praitheeshan, P., Pan, L., Yu, J., Liu, J., and Doss, R. (2019). Security analysis methods on ethereum smart contract vulnerabilities: a survey. *arXiv preprint arXiv:1908.08605*.

Qian, P., Liu, Z., He, Q., Huang, B., Tian, D., and Wang, X. (2022). Smart contract vulnerability detection technique: A survey. *arXiv preprint arXiv:2209.05872*.

Rameder, H., Di Angelo, M., and Salzer, G. (2022). Review of automated vulnerability analysis of smart contracts on ethereum. *Frontiers in Blockchain*, 5:814977.

Ressi, D., Spanò, A., Benetollo, L., Piazza, C., Bugliesi, M., and Rossi, S. (2024). Vulnerability detection in ethereum smart contracts via machine learning: A qualitative analysis. *arXiv preprint arXiv:2407.18639*.

Sharma, N. and Sharma, S. (2022). A survey of mythril, a smart contract security analysis tool for evm bytecode. *Indian J Natural Sci*, 13(75):39–41.

Torres, C. F., Iannillo, A. K., Gervais, A., and State, R. (2021). Confuzzius: A data dependency-aware hybrid fuzzer for smart contracts. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 103–119.

Wohrer, M. and Zdun, U. (2018). Smart contracts: security patterns in the ethereum ecosystem and solidity. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 2–8. IEEE.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32.

Zhou, H., Milani Fard, A., and Makanju, A. (2022). The state of ethereum smart contracts security: Vulnerabilities, countermeasures, and tool support. *Journal of Cybersecurity and Privacy*, 2(2):358–378.