Revista Eletrônica de Iniciação Científica em Computação, 2026, 24:1
ISSN: 3085-8461 • doi: 10.5753/reic.2026.6719

Artigo de Pesquisa/Research Paper

Avaliação da eficácia de analisadores de vulnerabilidades em contratos inteligentes de blockchains

Evaluating the effectiveness of vulnerability analyzers in blockchain smart contracts

Rafael Santa Rosa Alves [Universidade Estadual de Campinas | *r243472@dac.unicamp.br*]
Marco Amaral Henriques [Universidade Estadual de Campinas |*maah@unicamp.br*]

Resumo. A segurança de contratos inteligentes é um problema na blockchain Ethereum e todas as outras baseadas em EVM como a Hyperledger Besu, por exemplo. Este trabalho apresenta uma análise empírica da capacidade de detecção de vulnerabilidades em smart contracts Ethereum, com foco na evolução e eficácia das ferramentas utilizadas pelo conjunto de ferramentas *SmartBugs*. Em um primeiro experimento, foram executadas em 215 contratos reais coletados do Etherscan, revelando que 98% dos alertas gerados pelas ferramentas foram classificados como "outros", o que indica que a taxonomia DASP Top 10 encontra-se desatualizada frente ao cenário atual de desenvolvimento. Em outros experimentos, avaliamos a taxa de detecção real sobre uma base de contratos propositalmente vulneráveis, utilizando as versões 2.0.10 e 2.0.15 do *SmartBugs*. Além das ferramentas originais, foram incorporados novos analisadores estáticos e dinâmicos, foi adotada uma metodologia mais refinada de validação, baseada na localização exata da vulnerabilidade no código-fonte, e não apenas na correspondência nominal do tipo de falha. Os resultados mostram que, apesar da evolução entre as versões, ainda existem discrepâncias significativas entre as ferramentas que compõem o conjunto SmartBugs, com algumas apresentando melhorias substanciais na precisão enquanto outras mantêm desempenho abaixo do esperado. Os achados indicam que a classificação de vulnerabilidades utilizada nos estudos iniciais não reflete o estado atual do ecossistema, e que a ausência de padronização na validação dos achados ainda compromete análises comparativas.

Abstract. The security of smart contracts is a recurring issue not only in the Ethereum blockchain but also in other EVM-based networks such as Hyperledger Besu. This work presents an empirical analysis of the vulnerability detection capabilities of smart contract analysis tools, focusing on the evolution and effectiveness of the tools integrated into the *SmartBugs* framework. In the first experiment, 215 real contract samples collected from Etherscan were analyzed, revealing that 98% of the alerts generated by the tools were classified as "other", which indicates that the *DASP Top 10* taxonomy, used in previous studies, is outdated when compared to the current development landscape. In other experiments, we evaluated the actual detection rate on a dataset of intentionally vulnerable contracts, using versions 2.0.10 and 2.0.15 of *SmartBugs*. In addition to the original tools, new static and dynamic analyzers were incorporated, and a more refined validation methodology was adopted, based on the exact location of the vulnerability in the source code, rather than solely on nominal matching of the vulnerability type. The results show that, despite the evolution between versions, significant discrepancies still exist among the tools included in SmartBugs, with some showing substantial improvements in precision while others maintain performance below expectations. The findings indicate that the vulnerability classification used in the initial studies no longer reflects the current state of the ecosystem, and that the lack of standardization in the validation process still compromises comparative analyses.

Palavras-chave: Blockchain, contratos inteligentes, segurança, vulnerabilidades, DASP Top Ten

Keywords: Blockchain, Smart Contracts, security, vulnerabilities, DASP Top Ten

Recebido/Received: 20 October 2025 • **Aceito/Accepted:** 14 January 2026 • **Publicado/Published:** 23 January 2026

1 Introdução

A plataforma Ethereum popularizou o uso de contratos inteligentes (*smart contracts*) como uma forma descentralizada e autônoma de executar lógica computacional na blockchain. Aplicações como exchanges descentralizadas (DEXs), jogos, tokens e protocolos de finanças descentralizadas (DeFi) são hoje amplamente construídas sobre esses contratos, que movimentam bilhões de dólares diariamente. Entretanto, a complexidade e a imutabilidade dos smart contracts tornam a segurança um fator crítico: uma vulnerabilidade explorada pode levar à perda irreversível de fundos, como evidenciado por ataques históricos como o *DAO Hack* (2016) Mehar *et al*. [2017].

Diante desse cenário, a comunidade de segurança tem se mobilizado para desenvolver ferramentas capazes de detectar automaticamente vulnerabilidades em contratos inteligentes, utilizando técnicas de análise estática, simbólica e dinâmica Salzer and Di Angelo [2019] Pinna *et al*. [2019]. O ecossistema de ferramentas inclui soluções como Slither Feist *et al*. [2019], Mythril Mueller [2018], Manticore Mossberg *et al*. [2019] e Oyent JJ and Singh [2024], cada uma com abordagens e níveis de precisão distintos. Uma lista completa das ferramentas de análise automatizada configuradas no framework SmartBugs e utilizadas neste estudo, juntamente com suas versões específicas, será apresentada na Seção 3.1. Em paralelo, frameworks como o *SmartBugs* surgiram para facilitar a execução comparativa dessas ferramentas, padronizando a avaliação de seus resultados.

Apesar dos avanços nas técnicas de análise automatizada, ainda há lacunas na literatura no que diz respeito à evolução histórica da segurança em contratos inteligentes. A maioria dos estudos se concentra em análises pontuais ou avaliações de ferramentas sobre conjuntos fixos de contratos. Há pouca investigação sobre como essas ferramentas evoluem com o tempo — tanto em termos de cobertura de vulnerabilidades quanto em mudanças na própria taxonomia das falhas reconhecidas pela comunidade. Além disso, atualizações de ferramentas, inclusão de novos analisadores e correções em parsers podem alterar significativamente os resultados de experimentos anteriores, o que raramente é discutido nos estudos disponíveis.

Este artigo apresenta uma análise baseada em três experimentos distintos, ambos concebidos para permitir uma comparação com os resultados de um estudo conduzido em 2020 utilizando o framework SmartBugs Durieux *et al*. [2020b]. No primeiro experimento, analisamos um novo conjunto de contratos inteligentes recentemente verificados na rede Ethereum, coletados diretamente da plataforma Etherscan. A comparação com os achados de 2020 permite avaliar mudanças nas ocorrências das vulnerabilidades detectadas e inferir tendências evolutivas no desenvolvimento de contratos.

No segundo experimento, executamos novamente as ferramentas sobre o conjunto de contratos curado usado no estudo de 2020. Essa comparação visa avaliar a eficácia atual das ferramentas frente a casos conhecido. Um resultado inesperado surgiu nesse segundo experimento: uma versão mais atual `2.0.10` do SmartBugs apresentou uma queda significativa na taxa de detecção de vulnerabilidades, reduzindo de 48 vulnerabilidades identificadas para apenas 28. Entretanto, ao aplicarmos uma metodologia própria de reclassificação baseada na localização das vulnerabilidades nos *parses* gerados, essa mesma versão passou a detectar 67 vulnerabilidades, revelando que parte das falhas de identificação estava relacionada não à capacidade das ferramentas, mas às heurísticas de agregação e rotulagem adotadas pelo framework.

No terceiro experimento, estendemos a análise para o conjunto completo de 142 contratos curados disponíveis, indo além dos 69 contratos originais do estudo de 2020. Este experimento inclui, além da taxa de detecção, o registro de falso positivos (FP) e falso negativos (FN), elementos essencias para uma avaliação mais realista da utilidade prática das ferramentas.

As próximas seções estão organizadas da seguinte forma. A Seção 2 apresenta os fundamentos teóricos necessários para contextualizar o estudo, incluindo tipos de vulnerabilidades, abordagens de análise e o funcionamento do framework SmartBugs. A Seção 3 descreve a metodologia adotada e os três experimentos conduzidos. A Seção 4 reúne os resultados obtidos, organizados conforme cada experimento. Por fim, a Seção 5 apresenta as conclusões, limitações e possíveis direções para trabalhos futuros.

2 Fundamentação Teórica

2.1 Smart Contracts na Ethereum

Smart contracts são programas executados de forma determinística por todos os nós da blockchain Ethereum. Escritos majoritariamente na linguagem Solidity, esses contratos são compilados para bytecode da Ethereum Virtual Machine (EVM) Wood *et al*. [2014] e implantados em endereços específicos da rede. Uma vez implantado, o código é imutável, e sua execução é governada pelas regras do consenso da rede.

A lógica dos contratos pode incluir operações financeiras, controle de acesso, armazenamento de dados e interações com outros contratos. Como consequência, vulnerabilidades no código podem ser exploradas de forma irreversível. Diferente de sistemas tradicionais, não é possível aplicar correções após a descoberta de uma vulnerabilidade, o que intensifica a necessidade de métodos preventivos de análise.

2.2 Padrões ERC de Contratos na Ethereum

No ecossistema Ethereum, os padrões `ERC` (Ethereum Request for Comments) são especificações técnicas propostas pela comunidade para padronizar comportamentos e interfaces de contratos inteligentes. Esses padrões são submetidos e discutidos publicamente através do processo de *Ethereum Improvement Proposals* (EIPs) e sua adoção visa garantir interoperabilidade, previsibilidade e maior segurança na implementação de funcionalidades comuns.

Os padrões mais populares incluem o **ERC20** Chen *et al*. [2020], utilizado para representar tokens fungíveis, e o **ERC721** Casale-Brunet *et al*. [2021], voltado para tokens não fungíveis (NFTs). A aderência a esses padrões facilita a integração de contratos com carteiras, exchanges e ferramentas do ecossistema, além de reduzir a probabilidade de erros de implementação.

Entretanto, a conformidade com um padrão ERC não garante, por si só, a ausência de vulnerabilidades. Implementações mal feitas ou modificações não padronizadas ainda podem introduzir falhas críticas, como problemas de controle de acesso, chamadas inseguras e erros aritméticos.

2.3 Tipos de Vulnerabilidades

Diversas classes de vulnerabilidades têm sido documentadas na literatura e observadas em ataques reais Atzei *et al*. [2017]. Uma das taxonomias mais influentes para categorizá-las é a Decentralized Application Security Project (DASP) Top 10 NCC Group [2018], proposta em 2018, que organiza as vulnerabilidades mais recorrentes em contratos inteligentes Ethereum. Seguem as categorias incluídas nessa classificação.

- **Reentrância (Reentrancy):** possibilidade de chamadas recursivas a uma função antes da atualização completa do estado interno do contrato, como no caso do *DAO Hack*.
- **Controle de Acesso (Access Control):** falhas na definição ou implementação de mecanismos de controle de acesso, permitindo que usuários não autorizados executem funções críticas.
- **Aritmética (Arithmetic):** erros relacionados a operações aritméticas, como *overflows*, *underflows* ou divisões por zero, especialmente em versões antigas de Solidity.
- **Chamadas não verificadas (Unchecked Low Calls):** ausência de verificação do retorno de chamadas de baixo nível (`call`, `delegatecall`, `send`), o que pode mascarar falhas de execução.
- **Negação de serviço (Denial of service):** padrões de código que podem tornar funções inutilizáveis, seja por

consumo excessivo de gás ou lógica mal planejada.
- **Execução antecipada de transações (Front Running):** exploração de informações públicas sobre transações pendentes para se antecipar a operações críticas.
- **Manipulação temporal (Time Manipulation):** dependência de variáveis temporais parcialmente controladas pelos mineradores, como `block.timestamp`, pode ser explorada para manipular a lógica de contratos.
- **Endereço truncado (Short addresses):** falhas na validação do tamanho de endereços em chamadas de função, truncando valores passados via parâmetros.
- **Aleatoriedade Fraca (Bad Randomness):** uso de fontes de aleatoriedade previsíveis ou manipuláveis, que podem ser exploradas por participantes da rede para obter vantagem em decisões críticas do contrato.

Ainda há uma categoria adicional: a dos *Unknown Unknowns* que, na prática, funciona como um repositório conceitual para todas as demais vulnerabilidades, mostrando os limites das abordagens atuais de categorização e análise. Mesmo assim, essa taxonomia foi utilizada neste estudo como base para a categorização dos achados de vulnerabilidades, com o objetivo de permitir uma comparação padronizada com trabalhos anteriores.

2.4 Análises Automatizadas

A detecção precoce de vulnerabilidades em contratos inteligentes é essencial para garantir a segurança de aplicações descentralizadas. Nesse contexto, ferramentas de análise automatizada têm desempenhado um papel central, permitindo a inspeção sistemática de contratos sem intervenção humana direta. Essas ferramentas se diferenciam principalmente pela abordagem adotada, que influencia diretamente sua cobertura, desempenho e suscetibilidade a falsos positivos ou negativos.

2.4.1 Análise Estática

A análise estática consiste na inspeção do código-fonte de um programa sem sua execução, visando identificar vulnerabilidades, violações de boas práticas e potenciais comportamentos inesperados. No contexto de contratos inteligentes, essa abordagem é particularmente relevante devido à natureza imutável dos contratos após sua implantação, tornando a detecção precoce de falhas crítica para a segurança dos sistemas baseados em blockchain Grishchenko *et al*. [2018].

Ferramentas de análise estática aplicadas a contratos inteligentes operam, em geral, sobre uma linguagem de programação (como Solidity) ou diretamente sobre a representação intermediária do contrato (bytecode EVM). Elas buscam padrões de vulnerabilidades conhecidas, tais como problemas de aritmética, reentrância, manipulação temporal, entre outros.

Dentre as diversas ferramentas de análise estática desenvolvidas para o ecossistema Ethereum, destacam-se Slither Feist *et al*. [2019] e Mythril Mueller [2018]. A escolha dessas ferramentas neste trabalho deve-se ao fato de serem amplamente reconhecidas na literatura e na prática da indústria, além de oferecerem suporte contínuo a atualizações do compilador Solidity e uma ampla cobertura de padrões de vulnerabilidades.

O Slither é uma ferramenta baseada em análise semântica do código-fonte em Solidity, oferecendo uma variedade de detectores especializados para diferentes categorias de problemas, além de permitir extensibilidade com regras customizadas. Já o Mythril combina técnicas de análise estática e simbólica, analisando o bytecode dos contratos inteligentes para detectar vulnerabilidades mais complexas no fluxo de execução. Ambas as ferramentas são referências consolidadas na área e têm sido utilizadas em auditorias reais de projetos de alto impacto Kushwaha *et al*. [2022].

2.4.2 Análise Dinâmica

A análise dinâmica de contratos inteligentes consiste em observar o comportamento dos contratos durante sua execução real ou simulada Eshghie *et al*. [2021] a fim de identificar vulnerabilidades ou comportamentos inesperados que podem não ser visíveis apenas com a análise estática. Diferentemente dessa análise, que examina o código-fonte ou o bytecode sem executá-lo, a análise dinâmica envolve a interação ativa com o contrato, enviando transações, modificando estados e monitorando a resposta do sistema em tempo real.

Essa abordagem permite detectar vulnerabilidades que dependem de condições específicas de execução, como problemas de consumo excessivo de gás, ataques de reentrância, falhas de controle de acesso, e problemas de sincronia em interações multi-contrato. Ferramentas como o Echidna Grieco *et al*. [2020] e o Manticore Mossberg *et al*. [2019] são amplamente utilizadas nesse contexto. O Echidna realiza testes baseados em fuzzing direcionado: gera automaticamente milhares de inputs para contratos inteligentes em busca de violações de propriedades de segurança definidas pelo desenvolvedor. Já o Manticore combina fuzzing com execução simbólica para explorar profundamente diferentes caminhos possíveis de execução de um contrato, oferecendo uma análise ainda mais abrangente.

Uma vantagem da análise dinâmica é a capacidade de encontrar vulnerabilidades complexas que surgem apenas em cenários de execução específicos. No entanto, essa abordagem também apresenta desafios, como a dificuldade em garantir cobertura completa dos estados do contrato e a possibilidade de falsos negativos — vulnerabilidades que existem, mas não foram exploradas durante os testes.

A combinação de análise dinâmica com técnicas estáticas e formais é considerada uma prática recomendada para prover uma cobertura mais completa na identificação de vulnerabilidades em contratos inteligentes.

2.4.3 Análise Simbólica e Formal

- **Análise simbólica** consiste em explorar os diferentes caminhos de execução de um programa utilizando variáveis simbólicas em vez de valores concretos Wang *et al*. [2023]. Em vez de executar o código com entradas específicas, a ferramenta modela todas as possibilidades de execução simultaneamente, o que permite identificar vulnerabilidades relacionadas a estados raros ou a combinações de entradas específicas. Essa técnica é capaz de encontrar erros difíceis de serem detectados por métodos puramente sintáticos, como a execução indevida de funções críticas em condições específicas.
- **Verificação formal**, por outro lado, envolve a criação de modelos matemáticos dos contratos inteligentes e a aplicação de técnicas de lógica formal para provar que determinadas propriedades sempre serão verdadeiras (ou

detectar casos onde essas propriedades possam falhar). A verificação formal é considerada o padrão-ouro de confiabilidade, especialmente para contratos de alto valor, embora seu alto custo computacional e a necessidade de modelagem precisa representem desafios práticos.

Ferramentas como o Mythril aplicam análise simbólica para percorrer as possíveis execuções do bytecode dos contratos inteligentes, buscando vulnerabilidades complexas, como ataques de reentrância e problemas de integer overflow. Já plataformas como VeriSol Wang *et al*. [2020] e Certora Prover Bennour *et al*. [2024] focam na verificação formal, permitindo que desenvolvedores especifiquem invariantes que seus contratos devem obedecer, as quais são então matematicamente verificadas contra o código.

Embora promissoras, essas técnicas de verificação formal enfrentam limitações significativas na prática, como explosão do espaço de estados, necessidade de expertise em especificação formal e integração limitada com pipelines de desenvolvimento ágeis. Ainda assim, avanços recentes buscam tornar essas abordagens mais acessíveis para o desenvolvimento seguro de contratos inteligentes em escala.

Ferramentas híbridas têm ganhado destaque por combinar múltiplas técnicas. Um exemplo é o Mythril, que une análise simbólica com heurísticas estáticas, buscando um equilíbrio entre precisão e desempenho. Além disso, a evolução do ecossistema tem incentivado a integração de linters, analisadores de fluxo de controle e frameworks de teste baseados em fuzzing.

2.5 O Framework SmartBugs

O *SmartBugs* Ferreira *et al*. [2020] é um framework de código aberto desenvolvido para padronizar a execução de múltiplas ferramentas de análise sobre conjuntos de contratos inteligentes. Seu principal objetivo é permitir experimentos replicáveis e comparáveis, fornecendo uma infraestrutura controlada que abstrai detalhes da configuração e execução das ferramentas, garantindo consistência nos resultados.

O framework suporta a execução em lote de analisadores estáticos, simbólicos e dinâmicos, incluindo soluções populares como *Slither*, *Mythril*, *Manticore*, *Oyente* e *Securify*, além de ferramentas mais recentes incorporadas ao longo do tempo, como *Oyente+*, *Securify2* e *CPG Contract Checker* (CCC). Cada ferramenta é executada em contêineres isolados via Docker, assegurando que erros em um analisador não afetem os demais e permitindo que múltiplas versões de cada ferramenta sejam avaliadas simultaneamente.

Para lidar com a heterogeneidade das saídas geradas por cada analisador, o SmartBugs incorpora *parsers* específicos por ferramenta, responsáveis por extrair e normalizar os alertas detectados. Após essa etapa, um *parser* unificador agrega os resultados em um formato padronizado, permitindo que diferentes ferramentas sejam comparadas sob uma mesma estrutura de dados.

Além disso, o SmartBugs oferece conjuntos de testes com contratos vulneráveis documentados, como o dataset `SmartBugs-Wild` Durieux *et al*. [2020b] e o `SmartBugs-Curated`, com contratos cuidadosamente selecionados para cobrir diferentes classes de vulnerabilidades. Cada contrato inclui anotações sobre linhas vulneráveis, possibilitando a avaliação de métricas como taxa de detecção de verdadeiros positivos, falsos positivos e falsos negativos. Essa estrutura permite tanto a comparação direta entre ferramentas quanto a análise da evolução da detecção ao longo do tempo, como feito neste estudo ao comparar versões 2.0.10 e 2.0.15 do framework.

O SmartBugs também facilita a padronização dos resultados por meio de saídas estruturadas em JSON ou CSV, integráveis a scripts de processamento que consolidam vulnerabilidades detectadas por contrato e por ferramenta. Isso permite análises estatísticas detalhadas, mapeamento para taxonomias como a DASP Top 10 e comparação com achados históricos, viabilizando pesquisas longitudinais sobre segurança em contratos inteligentes.

Apesar de suas vantagens, o SmartBugs apresenta limitações. A execução de algumas ferramentas pode depender de versões específicas de compiladores ou bibliotecas externas, e certas vulnerabilidades modernas, especialmente de natureza econômica ou envolvendo interações entre múltiplos contratos, podem não ser detectadas. Além disso, a classificação de vulnerabilidades baseada apenas em nomes ou padrões estáticos limita a identificação de falhas contextuais, o que motivou a adaptação metodológica realizada neste estudo para análise baseada na posição das vulnerabilidades dentro do código.

Ao combinar isolamento, reprodutibilidade e suporte a múltiplas ferramentas, o SmartBugs se estabelece como uma infraestrutura essencial para avaliação rigorosa da eficácia de analisadores de contratos inteligentes, sendo amplamente utilizado em pesquisas acadêmicas e benchmarks de segurança.

2.6 Métricas de Avaliação: Precisão, Recall e F1-Score

A simples contagem de vulnerabilidades detectadas não é suficiente para avaliar a eficácia de uma ferramenta de análise. Uma ferramenta que reporta muitas vulnerabilidades pode, ao mesmo tempo, introduzir um grande volume de falsos positivos (FP), o que a tornaria impraticável do ponto de vista operacional. Por outro lado, uma ferramenta extremamente conservadora pode apresentar poucos falsos positivos, mas ao custo de deixar passar vulnerabilidades reais (FN), o que compromete diretamente a segurança do contrato. Para capturar esse trade-off entre detecção e ruído, foram adotadas três métricas clássicas da literatura de avaliação de sistemas de detecção: *Precisão*, *Recall* e *F1-Score*.

Precisão mede a proporção de alertas corretos em relação ao total de alertas emitidos pela ferramenta. Ela é definida como:

$$\text{Precisão} = \frac{VP}{VP + FP}$$

Uma precisão alta indica que a ferramenta gera poucos alarmes incorretos, reduzindo o esforço analítico do auditor ao filtrar resultados irrelevantes.

Recall (também chamado de *cobertura* ou *sensitivity*) representa a capacidade da ferramenta de detectar de fato as vulnerabilidades presentes no contrato:

$$\text{Recall} = \frac{VP}{VP + FN}$$

Um valor elevado de recall significa que a ferramenta é capaz de cobrir uma parcela significativa das vulnerabilidades

reais, ainda que à custa de ruído.

No contexto de segurança, há um dilema clássico: ferramentas com alto recall tendem a gerar muitos falsos positivos, enquanto ferramentas com alta precisão podem deixar passar vulnerabilidades críticas. Para conciliar essas duas perspectivas, utiliza-se o **F1-Score**, que é a média harmônica entre precisão e recall:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precisão} \cdot \text{Recall}}{\text{Precisão} + \text{Recall}}$$

A escolha do F1-Score como métrica agregadora se justifica especialmente neste estudo, pois não há interesse em privilegiar exclusivamente a redução de falsos positivos ou a maximização de detecção. O objetivo é capturar o equilíbrio operacional entre usabilidade prática (baixa taxa de ruído) e eficácia real de detecção. A inclusão dessas métricas, ausentes em estudos anteriores, permite uma comparação mais direta entre ferramentas e fornece um ponto de referência replicável para análises futuras.

3 Metodologia

Para investigar o estado atual da segurança automatizada de contratos inteligentes na Ethereum, foram conduzidos três experimentos independentes, com objetivos e conjuntos de dados distintos, mas utilizando a mesma infraestrutura de execução baseada no framework *SmartBugs*

3.1 Ferramentas Utilizadas

Ambas as análises foram conduzidas utilizando a infraestrutura do *SmartBugs* versão `2.0.10`, configurado com as seguintes ferramentas de análise: Honeybadger (commit ff30c9a); Maian (commit 4bab09a); Manticore (versão 0.3.7); Mythril (versão 0.24.7); Osiris (commit d1ecc37); Oyente (commit 480e725); Securify (versão 1); Slither (versão 0.10.4); Smartcheck (versão 1); Confuzzius (versão 0.0.1); Conkas (commit 4e0f256); Semgrep (commit c3a9f40); Sfuzz (commit 48934c0); Solhint (versão 3.3.8)[1]. Essas ferramentas foram executadas em ambiente Linux, utilizando contêineres Docker fornecidos pelo próprio SmartBugs para garantir reprodutibilidade e isolamento.

Durante o desenvolvimento deste trabalho, o *SmartBugs* passou por uma série de atualizações, evoluindo da versão `2.0.10`, utilizada na execução inicial do segundo experimento, até a versão `2.0.15`. Essa nova versão incorporou três ferramentas adicionais (Oyente+, Securify2 e CPG Contract Checker (CCC)) e introduziu atualizações relevantes em ferramentas já existentes, como Slither, Mythril, Semgrep e Solhint. Além disso, diversos bugs, especialmente relacionados ao parser e ao processo de normalização dos relatórios, foram corrigidos. Para avaliar o impacto dessas mudanças na capacidade de detecção, o segundo experimento foi replicado com a versão mais recente do framework, permitindo uma análise comparativa entre os resultados obtidos nas duas versões.

[1] As cinco últimas ferramentas listadas — Confuzzius, Conkas, Semgrep, Sfuzz e Solhint — não faziam parte do conjunto de ferramentas utilizado no *SmartBugs* versão `1.0.0` (2020).

3.2 Experimento 1: Análise da Ocorrências de Vulnerabilidades

Este experimento teve como foco principal examinar a ocorrência das vulnerabilidades detectadas por ferramentas automatizadas em contratos inteligentes implantados recentemente na rede Ethereum buscando identificar tendências, recorrências e possíveis mudanças no perfil das falhas ao longo do tempo. Naturalmente, os padrões observados dependem do conjunto analisado e, portanto, diferentes amostras de contratos podem produzir distribuições distintas de categorias e frequências de vulnerabilidades

Foram selecionados inicialmente os 500 contratos inteligentes mais recentemente verificados na Etherscan no dia 26 de abril de 2025 Esses contratos podem ser encontrados nos artefatos que acompanham este artigo [2]. Contratos verificados são aqueles cujo código-fonte foi disponibilizado publicamente e corresponde exatamente ao código implantado na blockchain, conferindo transparência e autenticidade Etherscan [2025].

A obtenção dos endereços dos contratos foi realizada por meio de uma técnica de extração automatizada da seção *Verified Contracts* do Etherscan. Em seguida, para cada endereço, utilizou-se a API pública da Etherscan para recuperar o código-fonte.

Para esta análise, foram considerados apenas contratos em Solidity e de arquivo único. Contratos verificados compostos por múltiplos arquivos foram descartados para simplificar o processamento e evitar inconsistências no ambiente de compilação. Após a filtragem, o número de contratos inteligentes foi reduzido para 215, os quais foram utilizados como base para este experimento.

3.3 Experimento 2: Replicação de Análise em Contratos com Vulnerabilidades Conhecidas

O segundo experimento teve como objetivo principal replicar um estudo publicado em 2020 Durieux *et al*. [2020b], utilizando os mesmos contratos com falhas conhecidas, mas aplicando as versões atualizadas das ferramentas disponíveis no SmartBugs. Com isso, buscou-se avaliar a eficácia atual das ferramentas na detecção de vulnerabilidades já documentadas.

Foi utilizado o conjunto de contratos vulneráveis conhecido como `smartBugs-curated` Durieux *et al*. [2020a], que contém contratos com falhas intencionais e previamente catalogadas, cobrindo múltiplas categorias da taxonomia DASP Top 10.

Foram utilizadas as mesmas ferramentas do Experimento 1 (mais as cinco extras que estão no final da lista – Seção 3.1) sobre os contratos vulneráveis. Os achados de vulnerabilidades foram comparados com os relatórios originais de 2020 para cada categoria. A precisão das ferramentas foi medida em termos de taxa de acerto por tipo de vulnerabilidade.

Este experimento foi executado em duas etapas distintas: inicialmente com a versão `2.0.10` do *SmartBugs*, replicando a configuração mais próxima possível do estudo de 2020, e posteriormente com a versão `2.0.15`, que inclui novas ferramentas, correções de parsing e atualizações. Os achados

[2] `https://github.com/regras/smart_contract_sec`

foram comparados com os relatórios originais de 2020 para cada categoria de vulnerabilidade, permitindo avaliar não apenas a evolução da precisão das ferramentas, mas também o impacto das mudanças internas do framework. A precisão foi medida em termos de taxa de acerto por tipo de vulnerabilidade em ambas as versões, possibilitando uma análise comparativa.

3.4 Extração e Processamento dos Resultados

Os resultados produzidos pelas ferramentas foram exportados para arquivo CSV, onde cada linha contém informações sobre a execução de cada contrato, incluindo:

- `filename`: identificador do contrato (caminho do arquivo);
- `toolid`: ferramenta utilizada para a análise;
- `toolmode`: modo de execução da ferramenta;
- `parser_version`: versão do parser utilizado;
- `runid`: identificador único da execução;
- `start`: tempo de início da execução;
- `duration`: tempo de execução da análise;
- `exit_code`: código de saída da execução (indicando sucesso ou falha);
- `findings`: vulnerabilidades encontradas;
- `infos`: informações adicionais sobre a execução;
- `errors`: erros encontrados durante a execução;
- `fails`: falhas no processo de análise.

No Experimento 2, os resultados foram inicialmente processados com scripts em Python para geração de estatísticas descritivas, identificação de contratos com múltiplas vulnerabilidades e agrupamento dos achados por tipo e frequência, a partir das colunas `findings`. Nesta primeira etapa, foi utilizada a tabela `vulnerabilities_mapping` do próprio *SmartBugs*, que associa os alertas emitidos pelas ferramentas aos rótulos de vulnerabilidade com base na nomenclatura empregada no estudo de 2020.

Em um segundo processamento, conduzido de forma independente, os mesmos resultados foram reinterpretados utilizando uma estratégia de mapeamento alternativa, proposta neste trabalho, que desconsidera o nome atribuído pela ferramenta e utiliza como critério primário a posição da ocorrência no código-fonte. Essa abordagem permitiu reclassificar achados que anteriormente eram descartados ou agrupados de forma imprecisa, gerando uma visão mais granular da cobertura real das ferramentas e revelando discrepâncias relevantes nos resultados produzidos pelo framework. Em casos como o Slither, a ausência de um mapeamento preciso para vulnerabilidades que detectavam, combinada com bugs no parser, resultava em zero detecções ou classificações errôneas. A metodologia alternativa, ao usar a localização exata da falha no código (linha/coluna) como chave de correspondência, permitiu resgatar vulnerabilidades que haviam sido corretamente identificados pelas ferramentas, mas perdidos no processo de agregação e rotulagem do framework.

3.5 Experimento 3: Avaliação Ampliada com o Dataset Curado (142 Contratos) e Inclusão de Falsos Positivos/Negativos

Embora o Experimento 2 tenha reproduzido fielmente a configuração do estudo original de 2020 com um conjunto de 69 contratos curados, o ecossistema analisado evoluiu significativamente nos últimos anos. A base de contratos do conjunto `SmartBugs-Wild` foi expandida para 142 amostras com vulnerabilidades documentadas manualmente, o que abre espaço para uma análise mais abrangente e representativa do estado atual das ferramentas de detecção.

Com o objetivo de registrar uma linha de base atualizada para futuras pesquisas, este terceiro experimento estende o conjunto analisado para os 142 contratos curados disponíveis em 2025. Diferente dos estudos anteriores, que se limitaram a relatar apenas a taxa de detecção, aqui também são incluídas estatísticas de *Falsos Positivos* (FP) e *Falsos Negativos* (FN), elementos críticos para uma avaliação mais realista da utilidade prática das ferramentas. Essa é uma lacuna ainda não preenchida na literatura, e o registro desses dados fornece um ponto de comparação valioso para trabalhos subsequentes.

Cada uma das ferramentas suportadas pelo *SmartBugs* foi executada individualmente sobre o conjunto completo de 142 contratos. As vulnerabilidades reais foram previamente rotuladas manualmente (totalizando 221 ocorrências confirmadas). Os achados das ferramentas foram então agregados e unificados para eliminar duplicações internas, resultando em 216 detecções automáticas únicas.

Esse experimento será mantido como base referencial para comparações futuras, servindo como linha de base estendida além dos 69 contratos ja analisados pela literatura.

4 Resultados

4.1 Ocorrência de Vulnerabilidades em Contratos Recentes - Experimento 1

A Tabela 1 apresenta a distribuição percentual dos 215 contratos afetados pelas principais categorias de vulnerabilidades, conforme definido pela taxonomia DASP Top 10. É importante destacar que um mesmo contrato pode apresentar múltiplas vulnerabilidades, pertencentes a diferentes categorias.

Tabela 1. Distribuição de contratos no Experimento 1 de acordo com as categorias de vulnerabilidades DASP Top 10

Tipo de Vulnerabilidade	Contratos afetados (%)
Unknown Unknowns	100,0
Arithmetic	20,9
Denial of Service	17,2
Access Control	9,3
Unchecked Low Calls	8,4
Time Manipulation	4,6
Front Running	0,0
Short Address	0,0
Bad Randomness	0,0
Reentrancy	0,0

Destaca-se a inexistência de vulnerabilidades *Front Run-*

ning, *Short Address*, *Bad Randomness* e *Reentrancy* no conjunto analisado. Existem duas interpretações possíveis para essa ausência: (i) essas classes de vulnerabilidade podem ter sido eliminadas por boas práticas e ferramentas de desenvolvimento modernas (como bibliotecas da OpenZeppelin e melhorias no compilador Solidity) ou (ii) pode haver limitações nos detectores atuais, que talvez não estejam mais sensíveis às variantes dessas falhas como eram anteriormente. A definição exata depende de uma análise manual aprofundada dos contratos, o que pode ser foco de um trabalho futuro.

O mapeamento dos achados para categorias padronizadas revelou uma limitação conceitual importante: aproximadamente 98% das vulnerabilidades apontadas não se enquadraram diretamente nas categorias do DASP Top 10, adotadas como base para o estudo. Essas vulnerabilidades, embora válidas e relevantes, foram classificadas como *Unknown Unknowns* por não apresentarem correspondência clara com as outras categorias definidas por esse modelo específico. Esses achados frequentemente englobam uma gama de problemas que vão além das clássicas falhas de exploração diretas da taxonomia. Muitas dessas detecções são mais por questões de padronização e boas práticas de codificação. Porém, pode haver erros de uso da linguagem Solidity ou até mesmo problemas na tentativa de melhorar a eficiência ou otimizar o uso do gás, resultando em novas vulnerabilidades relevantes, que não se enquadram perfeitamente nas categorias do DASP. Exemplos incluem `SOLIDITY_ERC20_APPROVE`, que sinaliza padrões problemáticos no uso da função *approve* de tokens ERC20, e `no_slippage_check`, uma falha crítica em aplicações de finanças descentralizadas (DeFi) que, se não mitigada, pode levar a perdas financeiras significativas para o usuário devido à derrapagem de preços.

Assim, embora o DASP englobe categorias amplas, ele não cobre diversas classes de vulnerabilidades hoje recorrentes no ecossistema, o que explica a discrepância, assumindo que as ferramentas não estejam equivocadas e que o DASP não esteja obsoleto.

Esse resultado evidencia uma desconexão entre a taxonomia DASP Top 10 (formulada em 2018) e o panorama atual de desenvolvimento de contratos inteligentes. Desde a popularização dos protocolos DeFi e o surgimento de ecossistemas mais complexos como zkSync, Base, Arbitrum e soluções L2 otimizadas para rollups, novas classes de vulnerabilidades passaram a surgir associadas a liquidez, atualizações de proxy, oráculos de preço e interações entre contratos — aspectos não contemplados no modelo original do DASP. Na prática, a ausência de uma taxonomia atualizada leva a uma sub-representação dessas falhas emergentes nas análises automatizadas, dificultando a criação de métricas comparáveis entre ferramentas e períodos históricos.

Diversos trabalhos recentes têm discutido alternativas de classificação. Uma linha de pesquisa busca alinhar as vulnerabilidades de smart contracts à terminologia da *Common Weakness Enumeration* (CWE) Staderini *et al*. [2020], permitindo interoperabilidade com bases de dados de segurança tradicionais, além de facilitar a adoção dessa taxonomia por ferramentas de auditoria já consolidadas na engenharia de software. Outra proposta é o uso de taxonomias específicas para blockchain, como o *Openscv* Vidal *et al*. [2024], que incorpora falhas de design e de lógica econômica, indo além de bugs puramente sintáticos ou de execução. Mais recentemente, surgem abordagens baseadas em *Large Language Models* que, em vez de mapear achados para categorias fixas, propõem classificações dinâmicas orientadas por similaridade semântica entre vulnerabilidades documentadas, permitindo a identificação de padrões mesmo em casos não previstos pelas taxonomias tradicionais.

Dessa forma, os resultados apresentados não apenas evidenciam o descompasso entre as ferramentas atuais e a taxonomia original, mas também reforçam a necessidade de um modelo mais adaptativo de classificação — potencialmente híbrido, combinando mapeamento para CWE, categorias econômicas específicas de DeFi e mecanismos de agrupamento automatizado assistidos por LLMs.

Além disso, a classificação obtida reflete especificamente o uso da taxonomia DASP Top 10. Caso taxonomias alternativas, como OWASP Smart Contract Top 10, fossem aplicadas, a distribuição entre categorias seria diferente, embora isso não altere o fato estrutural de que a maioria dos alertas das ferramentas não se encaixa plenamente em modelos tradicionais de classificação.

4.2 Contratos com Vulnerabilidades Conhecidas - Experimento 2

Ao todo, o conjunto `SmartBugs-curated` era originalmente documentado com 115 vulnerabilidades previamente catalogadas, distribuídas entre as diferentes categorias da taxonomia DASP Top 10 utilizadas no estudo de 2020. No entanto, durante a reprodução deste experimento, constatamos que a própria base de referência evoluiu ao longo dos últimos anos: novas vulnerabilidades foram oficialmente adicionadas à anotação dos mesmos contratos, elevando o total para 129 ocorrências conhecidas.

Dessa forma, embora os contratos analisados permaneçam exatamente os mesmos, o conjunto de vulnerabilidades de referência não é mais estático. A diferença observada nos resultados, portanto, não decorre de alterações no código-fonte ou na natureza intrínseca das falhas, mas sim de dois fatores distintos: (i) a evolução das ferramentas utilizadas e (ii) a atualização do mapeamento das vulnerabilidades reconhecidas pela comunidade.

É importante destacar que o conjunto SmartBugs Curated foi construído especificamente para cobrir as categorias clássicas do DASP Top 10, incluindo exemplos intencionais de cada classe. Essa natureza controlada explica a alta correspondência entre as vulnerabilidades anotadas e o DASP, em contraste com os contratos coletados no Experimento 1, que refletem práticas e padrões contemporâneos de desenvolvimento e não foram selecionados com base em taxonomias tradicionais.

A Tabela 2 resume a evolução na detecção de vulnerabilidades ao longo do tempo, considerando diferentes versões do SmartBugs e metodologias de processamento. Observa-se que, embora os contratos analisados permaneçam os mesmos do estudo original de 2020, o total de vulnerabilidades catalogadas aumentou de 115 para 129, refletindo a atualização da base de referência e o refinamento contínuo de vulnerabilidades pela comunidade.

Na execução com a versão `2.0.10` do SmartBugs utilizando o mapeamento padrão, a taxa de detecção geral caiu de

Tabela 2. Evolução na detecção das vulnerabilidades entre 2020 e 2025

Categoria	Total	v1.0.0	v2.0.10	v2.0.10*	v2.0.15*
Access Control	24	5	3	9	22
Arithmetic	22	19	8	20	22
Bad Randomness	34	0	0	4	34
Denial of Service	14	0	3	8	14
Front Running	7	2	3	3	6
Other	5	0	3	2	4
Reentrancy	8	7	5	6	8
Short Addresses	1	0	0	1	1
Time Manipulation	5	3	3	5	5
Unchecked Low Calls	9	9	3	9	9
Total	129	48	28	67	125

* indica que foram utilizados critérios de mapeamento baseados na posição da vulnerabilidade no código, e não apenas no nome atribuído pela ferramenta.

48 para 28 vulnerabilidades detectadas, evidenciando falhas nas heurísticas internas do framework ou limitações de compatibilidade com padrões antigos. Quando a mesma versão foi processada com a metodologia de reclassificação baseada na posição das vulnerabilidades (`v2.0.10*`), o número de detecções subiu para 67, mostrando que parte da perda de acurácia não se deve às ferramentas em si, mas ao modo como os achados eram agregados e rotulados.

A versão `2.0.15*`, que incorpora correções de parser, novas ferramentas (**Oyente+**, **Securify2** e **CPG Contract Checker**) e atualizações nas ferramentas já existentes, conseguiu identificar 125 das 129 vulnerabilidades, um aumento significativo em relação às execuções anteriores. Destaca-se, por exemplo, a ferramenta **Slither**, que na versão 2.0.10 não detectou vulnerabilidades, mas na versão 2.0.15 identificou 94 delas, evidenciando melhorias concretas na cobertura de análise.

A comparação entre as versões `2.0.10` e `2.0.15` na tabela 3 evidencia um comportamento assimétrico quanto à evolução das ferramentas de análise. Na versão `2.0.10`, apenas quatro ferramentas (*mythril*, *osiris*, *confuzzius* e *conkas*) apresentaram taxas de detecção superiores a 20%, com *conkas* liderando com 34,88%. A maioria permaneceu com 0% de cobertura, demonstrando baixa capacidade de identificação frente às 129 vulnerabilidades reais presentes no conjunto analisado.

Com a atualização para a versão `2.0.15`, observa-se uma mudança expressiva, com ferramentas anteriormente ineficazes apresentando desempenho significativamente superior. *Slither*, que não detectava nenhuma vulnerabilidade na versão anterior, passou a identificar 72,87% das falhas, emergindo como a ferramenta com melhor desempenho isolado. Da mesma forma, *solhint*, *ccc* e *smartcheck* exibiram resultados relevantes.

Ferramentas como *conkas*, *osiris* e *confuzzius* mantiveram estabilidade entre as versões, indicando maturidade ou estagnação em sua capacidade de detecção. Já *mythril* apresentou ganho marginal, sugerindo evolução incremental. Em contrapartida, ferramentas como *maian*, *manticore* e *securify* permaneceram com 0% de detecção em ambas as versões, demonstrando que a atualização do ambiente *SmartBugs* não implica, por si só, melhorias uniformes.

Esses resultados reforçam que a eficácia das ferramentas deve ser avaliada de forma individualizada e não apenas em função da versão do framework utilizado. A disparidade observada indica que a evolução do ecossistema de análise está acontecendo de maneira desigual, com algumas ferramentas avançando rapidamente enquanto outras permanecem completamente inoperantes frente às vulnerabilidades conhecidas.

Esses resultados reforçam que a evolução das ferramentas e o refinamento metodológico têm impacto direto na acurácia, sendo que a combinação de novas versões e estratégias de reclassificação é essencial para compreender corretamente a eficácia das análises automatizadas.

4.3 Contratos com Vulnerabilidades Conhecidas - Experimento 3

A Tabela 4 evidencia uma disparidade significativa entre as estratégias adotadas pelos analisadores. Ferramentas como *Slither* e *Solhint* apresentam altos valores de recall (0.810 e 0.733, respectivamente), indicando que conseguem sinalizar a maior parte das vulnerabilidades reais — porém, à custa de uma quantidade extremamente elevada de falsos positivos, o que se reflete diretamente em suas baixas precisões (0.093 e 0.022). Por outro lado, ferramentas como *Confuzzius* e *CCC* exibem um equilíbrio mais interessante entre detecção e ruído, com F1-scores de 0.381 e 0.412, sugerindo um comportamento mais alinhado a cenários de uso prático, onde a triagem manual de alertas possui custo real.

Para este experimento, o conjunto ampliado de 142 contratos contém 221 vulnerabilidades reais previamente anotadas, número que serve como referência direta para o cálculo de VP, FP e FN apresentado na Tabela 4.

Um ponto importante é que estes resultados não possuem referência direta na literatura anterior. Estudos como o de Thomas et al. (2020) Durieux *et al*. [2020b] reportaram apenas a cobertura bruta (VP), sem registrar explicitamente FP e FN em escala unificada. Portanto, esta análise representa um primeiro esforço sistemático de quantificação da relação sinal/ruído dessas ferramentas sob um mesmo conjunto curado e consolidado. A ausência de uma linha histórica impede comparações temporais, mas abre espaço para que futuros trabalhos utilizem esta tabela 4 como referência para monitorar a maturidade das ferramentas de segurança no contexto de contratos inteligentes.

Tabela 3. Comparação da taxa de cobertura de vulnerabilidades por ferramenta nas versões 2.0.10 e 2.0.15

Ferramenta	Real	Detec. (2.0.10)	Taxa (%)	Detec. (2.0.15)	Taxa (%)	Variação (%)
honeybadger	129	2	1.55	2	1.55	0.00
maian	129	0	0.00	0	0.00	0.00
manticor	129	0	0.00	0	0.00	0.00
mythril	129	38	29.46	39	30.23	+0.77
securify	129	0	0.00	0	0.00	0.00
slither	129	0	0.00	94	72.87	**+72.87**
smartcheck	129	0	0.00	42	32.56	+32.56
osiris	129	31	24.03	31	24.03	0.00
confuzzius	129	39	30.23	39	30.23	0.00
conkas	129	45	34.88	45	34.88	0.00
semgrep	129	0	0.00	21	16.28	+16.28
sfuzz	129	0	0.00	0	0.00	0.00
solhint	129	0	0.00	77	59.69	+59.69
ethlint	129	0	0.00	0	0.00	0.00
ccc	129	0	0.00	72	55.81	+55.81
securify2	129	0	0.00	0	0.00	0.00
oyente+	129	0	0.00	24	18.60	+18.60

Tabela 4. Cobertura de detecção por ferramenta com métricas de desempenho

Ferramenta	VP	FP	FN	Precisão	Recall	F1
honeybadger	2	35	219	0.054	0.009	0.015
maian	0	0	221	–	0.000	–
manticore	0	0	221	–	0.000	–
mythril	86	463	135	0.157	0.389	0.223
securify	0	0	221	–	0.000	–
slither	179	1735	42	0.093	0.810	0.166
smartcheck	122	1352	99	0.083	0.552	0.145
osiris	96	420	125	0.186	0.434	0.260
confuzzius	113	259	108	0.304	0.511	0.381
conkas	127	705	94	0.152	0.575	0.241
semgrep	39	1009	182	0.037	0.176	0.062
sfuzz	0	0	221	–	0.000	–
solhint	162	7108	59	0.022	0.733	0.043
ethlint	0	0	221	–	0.000	–
ccc	163	407	58	0.285	0.737	0.412
securify2	0	0	221	–	0.000	–
oyente+	78	234	143	0.250	0.352	0.293

Além da análise por ferramenta, também foi realizada a segmentação dos achados por categoria de vulnerabilidade seguindo a taxonomia DASP Top 10, o que permite observar não apenas quantas vulnerabilidades foram detectadas, mas quais tipos de vulnerabilidade recebem maior ou menor atenção das ferramentas modernas. A distribuição está apresentada na Tabela 5:

A distribuição revela que Unchecked Low-Level Calls, Reentrancy e Bad Randomness concentram a maior parte das vulnerabilidades reais, o que está em linha com incidentes históricos na Ethereum, como o já consolidado DAO Hack e diversas explorações envolvendo sorteios pseudoaleatórios e envio inseguro de fundos. Em contraste, categorias como *Short Addresses* e *Other* aparecem apenas de forma residual, reforçando que algumas classes de falhas presentes em benchmarks acadêmicos possuem baixa representatividade prática

Tabela 5. Distribuição de vulnerabilidades por categoria

Categoria	Total Real	Detectadas
Access Control	24	22
Arithmetic	23	23
Bad Randomness	34	34
Denial of Service	14	14
Front Running	7	6
Other	5	4
Reentrancy	31	30
Short Addresses	1	1
Time Manipulation	7	7
Unchecked Low-Level Calls	75	75
Total	**221**	**216**

no ecossistema real de contratos.

Vale destacar que a alta incidência de certas categorias não implica necessariamente maior cobertura. Em especial, vulnerabilidades como *Front Running* e *Time Manipulation*, embora menos frequentes, apresentam um padrão de subdetecção em diversas ferramentas, indicando que o desafio nesses casos está mais relacionado ao contexto de execução e semântica de transação do que à simples análise sintática do código.

5 Conclusão

Este estudo replicou e expandiu uma análise histórica da segurança de contratos inteligentes na rede Ethereum, considerando a evolução do framework SmartBugs e das ferramentas de análise disponíveis entre 2020 e 2025. A execução de experimentos em duas versões do SmartBugs, aliada à metodologia de reclassificação baseada na posição das vulnerabilidades, permitiu revelar limitações e avanços não perceptíveis em análises tradicionais baseadas apenas na nomenclatura dos alertas.

Os resultados demonstram que, embora a versão mais recente do SmartBugs (`2.0.15*`) tenha alcançado uma cobertura quase completa (125 de 129 vulnerabilidades detec-

tadas), versões anteriores apresentaram regressões significativas. Além disso, a evolução do próprio conjunto de vulnerabilidades catalogadas, de 115 para 129, evidencia que parte do desafio está relacionado à classificação correta das falhas. Outro resultado importante foi a observação de que 98% dos achados do Experimento 1 foram classificados como "outros" reforça que a DASP Top 10 está desatualizada e não reflete integralmente o panorama atual de vulnerabilidades em contratos inteligentes.

Este estudo evidencia ainda que métricas agregadas, como número médio de vulnerabilidades detectadas por contrato, podem mascarar problemas metodológicos e limitações de ferramentas. A análise detalhada por versão, ferramenta e categoria revelou diferenças significativas entre execuções e destacou o impacto das novas ferramentas e correções implementadas.

Por fim, o Experimento 3 introduziu uma contribuição inédita ao estabelecer uma linha de base estendida com 142 contratos curados, incluindo o registro explícito de falsos positivos e falsos negativos por ferramenta. Essa etapa não apenas amplia a validade externa dos resultados anteriores, mas também cria um ponto de ancoragem para comparações longitudinais futuras, permitindo que evoluções do ecossistema possam ser medidas com maior precisão e menos ambiguidade interpretativa.

Como trabalhos futuros, planeja-se:

- Inclusão de ferramentas baseadas em Large Language Models (LLMs) para ampliar a cobertura e precisão na identificação de vulnerabilidades;
- Expansão da análise para contratos multi-arquivo e novas amostras mais recentes da rede Ethereum;
- Análise mais profunda de casos de vulnerabilidades não detectadas por nenhuma ferramenta, buscando compreender lacunas persistentes na análise automatizada;
- Exploração da construção de datasets mais robustos e escaláveis para avaliação de ferramentas de análise de vulnerabilidades.

Tais medidas visam tornar os estudos de segurança em contratos inteligentes mais precisos, comparáveis e relevantes para o avanço seguro da tecnologia blockchain, contribuindo para uma avaliação longitudinal mais consistente das ferramentas automatizadas.

Declarações complementares

Financiamento

Trabalho parcialmente financiado pelo CPQD - projeto OP72003_Tokenizacao-de-Ativos-Ambientais-FIN TERMOPERNAMBUCO S.A. 03.795.050/0001-09.

Contribuições dos autores

Rafael Santa Rosa Alves contribuiu com a Concepção da metodologia, Coleta e curadoria dos contratos, Execução dos experimentos, Análise e interpretação dos resultados, Redação inicial do manuscrito e Visualização dos dados. Marco A. Henriques contribuiu com Supervisão, Validação das interpretações, Ajustes nas argumentações e Revisão crítica e redação final do manuscrito. Ambos os autores leram e aprovaram a versão final do trabalho.

Conflitos de interesse

Os autores declaram não ter nenhum conflito de interesses.

Disponibilidade de dados e materiais

Os conjuntos de dados e softwares gerados durante o estudo atual estão disponíveis em `https://github.com/regras/smart_contract_sec`.

Referências

Atzei, N., Bartoletti, M., and Cimoli, T. (2017). A survey of attacks on ethereum smart contracts (sok). *Principles of Security and Trust (POST)*, 10204:164–186. DOI: 10.1007/978-3-662-54455-6_8.

Bennour, I. E., Wannes, M. H., Ghiss, M., Braham, M., Lahbib, A., Habib, N., and Ribeiro, H. (2024). Enhancing dapp supply chain with verified smart contracts: A case study on the olive-oil industry. In *2024 IEEE/ACS 21st International Conference on Computer Systems and Applications (AICCSA)*, pages 1–8. DOI: 10.1109/AICCSA63423.2024.10912547.

Casale-Brunet, S., Ribeca, P., Doyle, P., and Mattavelli, M. (2021). Networks of ethereum non-fungible tokens: A graph-based analysis of the erc-721 ecosystem. In *2021 IEEE International Conference on Blockchain (Blockchain)*, pages 188–195. DOI: 10.1109/Blockchain53845.2021.00033.

Chen, W., Zhang, T., Chen, Z., Zheng, Z., and Lu, Y. (2020). Traveling the token world: A graph analysis of ethereum erc20 token ecosystem. In *Proceedings of The Web Conference 2020*, WWW '20, page 1411–1421, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3366423.3380215.

Durieux, T., Ferreira, H., Abreu, R., and State, R. (2020a). SmartBugs-curated: Dataset of vulnerable ethereum smart contracts. `https://github.com/smartbugs/smartbugs-curated`.

Durieux, T., Ferreira, J. F., Abreu, R., and Cruz, P. (2020b). 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.

Eshghie, M., Artho, C., and Gurov, D. (2021). Dynamic vulnerability detection on smart contracts using machine learning.

Etherscan (2025). Verified Contracts - Etherscan. `https://etherscan.io/contractsVerified`.

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. DOI: 10.1109/WETSEB.2019.00008.

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.

Grieco, G., Song, W., Cygan, A., Feist, J., and Groce, A. (2020). Echidna: effective, usable, and fast fuzzing for smart contracts. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2020, page 557–560, New York, NY, USA. Association for Computing Machinery. DOI:

10.1145/3395363.3404366.

Grishchenko, I., Maffei, M., and Schneidewind, C. (2018). Foundations and tools for the static analysis of ethereum smart contracts. In Chockler, H. and Weissenbacher, G., editors, *Computer Aided Verification*, pages 51–78, Cham. Springer International Publishing.

JJ, L. and Singh, K. (2024). Enhancing oyente: four new vulnerability detections for improved smart contract security analysis. *International Journal of Information Technology*, 16(6):3389–3399.

Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., and Lee, H.-N. (2022). Ethereum smart contract analysis tools: A systematic review. *IEEE Access*, 10:57037–57062. DOI: 10.1109/ACCESS.2022.3169902.

Mehar, M., Shier, C., Giambattista, A., Gong, E., Fletcher, G., Sanayhie, R., Kim, H. M., and Laskowski, M. (2017). Understanding a revolutionary and flawed grand experiment in blockchain: The dao attack. *Journal of Cases on Information Technology*, 21(1):19–32.

Mossberg, M., Manzano, F., Hennenfent, E., Groce, A., Grieco, G., Feist, J., Brunson, T., and Dinaburg, A. (2019). Manticore: A user-friendly symbolic execution framework for binaries and smart contracts.

Mueller, B. (2018). Smashing ethereum smart contracts for fun and real profit. *HITB SECCONF Amsterdam*, 9(54):4–17.

NCC Group (2018). Decentralized application security project (dasp) top 10. `https://www.dasp.co/`.

Pinna, A., Ibba, S., Baralla, G., Tonelli, R., and Marchesi, M. (2019). A massive analysis of ethereum smart contracts empirical study and code metrics. *IEEE Access*. DOI: 10.1109/ACCESS.2019.2921936.

Salzer, G. and Di Angelo, M. (2019). A survey of tools for analyzing ethereum smart contracts. DOI: 10.1109/DAPP-CON.2019.00018.

Staderini, M., Palli, C., and Bondavalli, A. (2020). Classification of ethereum vulnerabilities and their propagations. In *2020 Second International Conference on Blockchain Computing and Applications (BCCA)*, pages 44–51. DOI: 10.1109/BCCA50787.2020.9274458.

Vidal, F. R., Ivaki, N., and Laranjeiro, N. (2024). Openscv: An open hierarchical taxonomy for smart contract vulnerabilities. *Empirical Software Engineering*, 29(4):101.

Wang, Y., Lahiri, S. K., Chen, S., Pan, R., and Dillig, I. (2020). Formal verification of workflow policies for smart contracts in azure blockchain. In *International Conference on Verified Software: Theories, Tools, and Experiments*, pages 230–250. Springer. DOI: 10.1007/978-3-030-41600-3_7.

Wang, Y., Sheng, S., and Wang, Y. (2023). *A Systematic Literature Review on Smart Contract Vulnerability Detection by Symbolic Execution*, pages 226–241. DOI: 10.1007/978-981-99-8101-4_16.

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