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

Rafael Santa Rosa Alves[1], **Marco Amaral Henriques**[1]

[1]Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas
Campinas – São Paulo – Brasil

r243472@dac.unicamp.br, maah@unicamp.br

Abstract. *Smart contract security remains a critical challenge on the Ethereum blockchain. This paper investigates the evolution of automated security analysis tools via two experiments leveraging the SmartBugs framework. The first analyzes 215 recently verified Etherscan contracts, focusing on current vulnerability distribution. The second replicates a 2020 study, using its curated dataset of known vulnerabilities but with updated tool versions. Results indicate a lag in the DASP Top 10 taxonomy, and a concerning drop in detection accuracy on the curated dataset—from 41.7% to 24.3%, raising doubts on the tools' progress.*

Resumo. *A segurança de contratos inteligentes continua sendo um desafio na blockchain Ethereum. Este artigo investiga a evolução de ferramentas de análise de segurança por meio de dois experimentos com a estrutura SmartBugs. O primeiro analisa 215 contratos do Etherscan verificados recentemente, focando nas vulnerabilidades detectadas. O segundo replica um estudo de 2020, usando o mesmo conjunto de contratos com vulnerabilidades, mas com ferramentas atualizadas. Resultados indicam defasagem da taxonomia DASP Top 10 e uma queda na precisão de detecção (de 41,7% para 24,3%), levantando dúvidas sobre o real progresso das ferramentas.*

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 estão de fato evoluindo em sua capacidade de encontrar vulnerabilidades, especialmente considerando mudanças na linguagem Solidity, na popularização de bibliotecas que fornecem contratos pré-auditados e padronizados, como a OpenZeppelin [OpenZeppelin], e na profissionalização do desenvolvimento de contratos.

Este artigo apresenta uma análise baseada em dois 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: mesmo com versões atualizadas das ferramentas, a taxa de detecção de vulnerabilidades caiu de 41,7% para 24,3%.

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 de baixo nível 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 que visa padronizar a execução de múltiplas ferramentas de análise sobre um mesmo conjunto de contratos inteligentes. Ele permite a análise em lote, a extração dos resultados em formato estruturado (JSON) e a comparação entre ferramentas. Além disso, o projeto oferece conjuntos de testes com contratos vulneráveis documentados (como o dataset `SmartBugs-Wild` [Durieux et al. 2020b]) e suporte a múltiplos analisadores.

Ao utilizar o SmartBugs, pesquisadores podem conduzir experimentos controlados, replicáveis e comparáveis, o que contribui diretamente para uma avaliação mais rigorosa da segurança dos contratos inteligentes e da eficácia das ferramentas automatizadas.

3. Metodologia

Para investigar o estado atual da segurança automatizada de contratos inteligentes na Ethereum, foram conduzidos dois 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*, 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.

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.

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

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

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

cujo código-fonte foi disponibilizado publicamente e corresponde exatamente ao código implantado na blockchain, conferindo transparência e autenticidade [Etherscan].

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.

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 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` e `exit code`. As saídas das ferramentas, que indicam padrões potencialmente problemáticos no código foram posteriormente mapeadas para as categorias da DASP Top 10. Esse mapeamento permitiu analisar a distribuição percentual das vulnerabilidades segundo uma taxonomia reconhecida, favorecendo a comparação com o estudo anterior.

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 Running*, *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. Apesar de muitas dessas detecções serem mais por questões de padronização e boas práticas de codificação, erros de uso da linguagem Solidity, ou até mesmo eficiência e otimização do gás, ainda há detecções que apontam para vulnerabilidades relevantes, mas 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.

Isso reflete o caráter dinâmico das vulnerabilidades e sua classificação, já que novos tipos surgem a todo momento e não podem ser classificadas adequadamente pela taxonomia vigente. Isso demonstra que a taxonomia de 2018 já está obsoleta, e precisa se atualizar para o novo conjunto de vulnerabilidades que estão surgindo. Diante da defasagem da DASP Top 10, outras abordagens para classificação de vulnerabilidades em contratos inteligentes têm sido propostas na literatura. Incluem-se tentativas de mapeamento para a Common Weakness Enumeration (CWE) [Staderini et al. 2020], uma lista mais abrangente de tipos de falhas de software que pode ser adaptada ao contexto blockchain, bem como taxonomias específicas desenvolvidas em estudos acadêmicos mais recentes como o Openscv [Vidal et al. 2024].

4.2. Contratos com Vulnerabilidades Conhecidas - Experimento 2

A Tabela 2 apresenta a comparação da capacidade de detecção de vulnerabilidades por ferramentas automatizadas de análise entre o estudo conduzido em 2020 e os resultados obtidos neste trabalho em 2025. Para cada categoria, o valor apresentado representa a proporção de vulnerabilidades corretamente detectadas em relação ao total de ocorrências conhecidas daquela categoria no conjunto de contratos.

Ao todo, o conjunto `SmartBugs-curated` contém 115 vulnerabilidades previamente catalogadas, distribuídas entre as diferentes categorias da taxonomia DASP Top 10. Importante destacar que, neste experimento, os contratos analisados em ambos os casos são exatamente os mesmos, contendo falhas previamente conhecidas e catalogadas. A diferença observada, portanto, não decorre de alterações nos contratos ou na natureza das vulnerabilidades, mas sim da evolução — ou regressão — das ferramentas utilizadas.

Observa-se uma queda acentuada na taxa de detecção para categorias como vulnerabilidades *Arithmetics* (de 86,4% para 36,4%) e *Unchecked Low Calls* (de 75,0% para 25,0%). Tais reduções podem indicar mudanças nas heurísticas internas das ferramentas ou perda de compatibilidade com padrões antigos. Por outro lado, a detecção de vulnerabilidades de negação de serviço (*denial of service*), ausente no estudo anterior, foi significativamente melhor nesta nova análise, sugerindo avanços em áreas anteriormente pouco cobertas.

No geral, contudo, o total de vulnerabilidades detectadas caiu de 48 (41,7%) para 28 (24,3%) dentre as 115 existentes. Esse resultado levanta preocupações quanto à robustez e consistência das ferramentas atuais ao lidar com casos já conhecidos, reforçando a necessidade de avaliação contínua da eficácia das abordagens automatizadas.

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

Tipo	Quantidade presente	Detectadas 2020		Detectadas 2025	
Unknown Unknowns	3	3	(100,0%)	3	(100,0%)
Reentrancy	8	7	(87,5%)	5	(62,5%)
Time Manipulation	5	3	(60,0%)	3	(60,0%)
Denial of Service	7	0	(0,0%)	3	(42,9%)
Front Running	7	2	(28,6%)	3	(42,9%)
Arithmetic	22	19	(86,4%)	8	(36,4%)
Unchecked Low Calls	12	9	(75,0%)	3	(25,0%)
Access Control	19	5	(26,3%)	3	(15,8%)
Bad Randomness$^{(*)}$	31	0	(0,0%)	0	(0,0%)
Short addresses$^{(*)}$	1	0	(0,0%)	0	(0,0%)
Total	115	48	(41,7%)	28	(24,3%)

$^{(*)}$ Vulnerabilidades não detectadas pelas ferramentas.

5. Conclusão

Este estudo replicou e expandiu uma análise da segurança de contratos inteligentes na rede Ethereum, utilizando a infraestrutura do SmartBugs com um conjunto ampliado de ferramentas de análise. Embora o número médio de achados por contrato tenha aumentado, essa métrica isolada deve ser interpretada com cautela, considerando a ampliação do conjunto de analisadores e as limitações na categorização das vulnerabilidades.

A análise revelou padrões estruturais consistentes, como a predominância de contratos baseados em ERC20 e a frequência de classes de falhas como *Access Control*, *Arithmetic* e *Denial of Service*. Contudo, a replicação de um experimento anterior com contratos intencionalmente vulneráveis revelou um dado preocupante: a taxa de detecção geral caiu de 41,7% para 24,3%, apesar da maior disponibilidade de ferramentas. Embora as razões exatas para essa regressão não tenham sido totalmente determinadas neste estudo, essa queda sugere que avanços em cobertura e desempenho não se traduziram diretamente em maior eficácia na detecção de vulnerabilidades reais e será um ponto crucial para investigações futuras

Uma limitação observada neste contexto é que o dataset utilizado, por ser relativamente pequeno (contendo 115 vulnerabilidades previamente catalogadas), pode ter seus resultados impactados significativamente por pequenas alterações nas ferramentas de detecção. A falta de um dataset mais abrangente e atualizado é um desafio conhecido na área e alguns pesquisadores estão surgindo com propostas para corrigir isso, por meio de datasets construídos com o auxílio de Large Language Models (LLMs), visando oferecer bases mais robustas e escaláveis para a avaliação de ferramentas [Chen et al. 2025].

Como trabalhos futuros, pretende-se aprimorar a classificação dos achados, possibilitar a análise de contratos multi-arquivo, expandir a amostragem e investigar qualitativamente os motivos da regressão nas taxas de detecção. Tais medidas visam tornar os estudos de segurança em contratos inteligentes mais precisos, comparáveis e relevantes para o avanço seguro da tecnologia blockchain.

Agradecimentos

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

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.

Bennour, I., Wannes, M., and Ghiss, M. (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)*. IEEE.

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.

Chen, J., Shen, Y., Zhang, J., Li, Z., Grundy, J., Shao, Z., Wang, Y., Wang, J., Chen, T., and Zheng, Z. (2025). Forge: An llm-driven framework for large-scale smart contract vulnerability dataset construction. *arXiv preprint arXiv:2506.18795*.

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.

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. 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.

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.

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.

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/`.

OpenZeppelin. Openzeppelin contracts. `https://openzeppelin.com/contracts/`.

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*.

Salzer, G. and Di Angelo, M. (2019). A survey of tools for analyzing ethereum smart contracts.

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.

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.

Wang, Y., Sheng, S., and Wang, Y. (2023). *A Systematic Literature Review on Smart Contract Vulnerability Detection by Symbolic Execution*, pages 226–241.

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