International Journal of Engineering Research and Modern Education (IJERME)
Impact Factor: 7.018, ISSN (Online): 2455 - 4200
(www.rdmodernresearch.com) Volume 7, Issue 2, 2022

COMPARING AUTOMATED DETECTION OF ACCESS CONTROL, ARITHMETIC, AND REENTRANCY VULNERABILITIES IN CURATED ETHEREUM SMART CONTRACTS

Divya Vishwanath* & Paul Johnson Asamoah**
* Janardhan Rai Nagar Rajasthan Vidyapeeth, Udaipur, Rajasthan, India
** Power Life University College, Accra, Ghana, West Africa

Cite This Article: Divya Vishwanath & Paul Johnson Asamoah, "Comparing Automated Detection of Access Control, Arithmetic, and Reentrancy Vulnerabilities in Curated Ethereum Smart Contracts", International Journal of Engineering Research and Modern Education, Volume 7, Issue 2, Page Number 87-94, 2022.
Copy Right: © IJERME, 2022 (All Rights Reserved). This is an Open Access Article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract:

This analysis compares automated vulnerability detection in Ethereum smart contracts to inform tool selection for security reviews across jurisdictions. It uses published results from a curated benchmark released in 2020, comprising 27 observations across nine tools and three vulnerability categories. Descriptive comparisons measure matched detections relative to annotated vulnerabilities within each category. Mean detection across tools was 8.2% for access control, 21.7% for arithmetic, and 47.2% for reentrancy. Mythril achieved the highest arithmetic coverage at 68.2%, while Slither led reentrancy detection at 87.5% and shared the highest access control coverage at 21.1%. No tool led across all categories, indicating that aggregate rankings can conceal differences relevant to audit planning. These results describe benchmark coverage without identifying causal mechanisms or establishing performance beyond the selected annotations. The contribution is a focused secondary comparison that makes category specific detection differences explicit. For international auditing practice, the evidence supports evaluating coverage against relevant vulnerability classes and reporting the scope of automated checks alongside their results.

Key Words: Automated Analysis; Benchmark Evaluation; Ethereum; Smart Contract Security; Vulnerability Detection

1. Introduction:

Smart contracts enable parties to execute transactions through code, but errors in that code can expose assets and disrupt intended operations. Automated analysis tools help identify vulnerabilities before they are exploited. Their value depends on which weaknesses they detect and which they leave unrecognized. For developers and auditors, the central decision is therefore not only whether to use automated analysis, but which tools provide relevant coverage for the vulnerability classes under review. Luu et al., 2016, demonstrate the security consequences of flaws in Ethereum contract execution and introduce symbolic execution as one approach to identifying them.

Vulnerability detection is shaped by the interaction between contract behavior and the analytical properties of each tool. Access control analysis requires attention to authorization conditions. Arithmetic analysis concerns numerical operations and their possible outcomes. Reentrancy analysis requires attention to calls and state changes across an execution sequence. These categories represent different security problems, so a tool's performance in one category does not necessarily establish its performance in another. Atzei et al., 2017, distinguish these vulnerability mechanisms in their analysis of attacks on Ethereum contracts.

Program analysis provides the conceptual basis for comparing these outcomes. Tools use different representations and procedures to examine possible contract behavior. Feist et al., 2019, describe how Slither uses an intermediate representation to support analyses such as data flow and tracking of information through contract code. Chen et al., 2019, show that incomplete identification of control flows can constrain analysis, while execution traces can supply information absent from static analysis. These findings suggest that detection may depend on the relationship between a vulnerability pattern and the behavior represented by a tool. They do not establish the cause of differences in the present benchmark results.

Comparative evaluation also depends on the data and criteria used to assess tools. Ghaleb and Pattabiraman, 2020, evaluated static analysis tools using injected vulnerabilities and found both missed bugs and false positives. Ren et al., 2021, show that differences in datasets and evaluation settings can alter conclusions about tool performance. Together, these studies indicate that a detection result must be interpreted in relation to the benchmark, vulnerability category, and outcome measure. A tool that matches annotated vulnerabilities may still produce incorrect warnings, so detection coverage alone cannot establish overall accuracy.

The closest source for this analysis is the benchmark reported by Durieux et al., 2020. Their evaluation applied nine tools to a curated set of 69 annotated vulnerable contracts and to a larger collection of Ethereum contracts. The original work already reported differences in detection across tools and vulnerability categories. The present analysis uses its published category results to provide a focused descriptive comparison of access

control, arithmetic, and reentrancy. Its contribution is the organization and interpretation of those existing results around three category specific research questions. It does not provide an independent benchmark or establish a previously undocumented detection pattern.

This analysis measures detection as the proportion of annotated vulnerabilities in a category matched by a tool. The measure makes results comparable across tools within the same category because each tool uses the same category denominator. It does not identify which individual vulnerabilities were matched, measure false positive rates, or show whether tools detect the same cases. Differences between categories must also be interpreted cautiously because the annotated cases and denominators differ. The selected contracts form a curated benchmark, rather than a random sample of all Ethereum contracts.

This study aims to compare automated detection of access control, arithmetic, and reentrancy vulnerabilities in curated Ethereum smart contracts. The research question asks how the proportion of annotated vulnerabilities detected varies across the nine tools within each category. The specific objectives are to compare tool detection proportions for access control, assess the corresponding variation for arithmetic, and examine the variation for reentrancy. The analysis uses 27 tool and category observations from the published benchmark and applies descriptive comparisons of matched counts and proportions.

The results show that no single tool has the highest reported detection proportion in all three categories. This finding supports a category specific approach to interpreting tool coverage within the benchmark. For auditors working with Ethereum contracts across jurisdictions, the comparison can inform the choice of checks to examine further, particularly where authorization, arithmetic, or reentrancy risks are relevant. Transfer to other contracts and tool versions remains an empirical question because the analysis does not measure performance outside the selected benchmark.

2. Theory, Conceptual Framework and Research Questions:

Automated analysis of Ethereum smart contracts rests on program analysis theory. A tool represents contract code, examines selected execution paths or dependencies, and compares them with rules that describe unsafe behavior. Detection depends on the relationship between the vulnerability pattern and the properties the tool can analyze. Luu et al., 2016, demonstrate this logic through symbolic execution, which examines possible contract behaviors to identify security weaknesses.

This analysis treats vulnerability category as the independent construct and the proportion of annotated vulnerabilities detected as the dependent construct. For each tool and category, the detection proportion equals matched detections divided by the number of annotated vulnerabilities in that category. The comparison covers nine tools and three vulnerability categories in a curated benchmark. Durieux et al., 2020, show that detection differs across tools and vulnerability categories, providing an empirical basis for examining these differences.

The framework expects detection proportions to vary because each vulnerability category requires a tool to recognize a different pattern of contract behavior. Detection may improve when the tool's analysis matches the relevant pattern. Differences in the annotated examples and the rules implemented by each tool can also produce variation. These mechanisms guide interpretation; the available aggregate data do not isolate their individual effects. The arrows in the framework represent an analytical association between vulnerability category and detection proportion.

Access control vulnerabilities concern restrictions on who can execute sensitive contract operations. Their detection requires analysis of authorization conditions and the relationships between callers and protected operations. Tsankov et al., 2018, explain how security analysis can examine dependencies and check contract behavior against specified security properties. A tool may detect an annotated weakness when its rules capture the relevant authorization pattern, while a different or unsupported pattern may remain undetected.

RQ1: How does the proportion of annotated access control vulnerabilities detected vary across the nine analysis tools?

Arithmetic vulnerabilities arise when numerical operations produce unintended results. Their detection depends on whether an analysis can identify the conditions under which a calculation violates the intended numerical range. Ferreira Torres et al., 2018, show that symbolic execution and data tracing can be combined to identify integer weaknesses. This approach provides a reason to examine arithmetic detection separately from access control detection, while allowing the observed proportions to differ across tools.

RQ2: How does the proportion of annotated arithmetic vulnerabilities detected vary across the nine analysis tools?

Reentrancy vulnerabilities involve a sequence in which an external call permits a contract to be entered again before an earlier operation has completed its intended state changes. Detection therefore depends on recognizing relationships among calls and state updates. Atzei et al., 2017, identify reentrancy as a distinct form of smart contract vulnerability, while Feist et al., 2019, describe how program analysis can examine data flows and contract behavior. The available observations allow comparison of detection proportions, but a zero proportion alone does not establish whether a tool lacks support for the category.

RQ3: How does the proportion of annotated reentrancy vulnerabilities detected vary across the nine analysis tools?

The research questions are descriptive and permit similar, different, or inconclusive patterns. The available data report aggregate counts rather than the identity of each vulnerability detected by each tool. The analysis therefore compares reported proportions without treating repeated observations as independent evidence for a causal effect.

3. Data:

This section establishes the provenance and construction of the analytical dataset. It defines the observed measures and the descriptive comparisons used to answer the research questions.

3.1 Data Sources, Population, and Sample Construction:

The analysis uses the curated benchmark results reported by Durieux et al., 2020, and reproduced in the SmartBugs results repository. The source reports automated analysis results for annotated Ethereum smart contracts written in Solidity. The repository identifies the underlying curated benchmark as containing 69 contracts. The selected data are the category level counts in the repository's curated results table.

The target population comprises automated vulnerability detection results for Ethereum smart contracts. The accessible sampling frame is narrower: the nine tools and annotated vulnerability categories represented in the published curated benchmark. The analytical sample retains access control, arithmetic, and reentrancy because these are the three categories addressed by the research questions. The geographic scope is the Ethereum network rather than a specific country. The observations represent a benchmark released in 2020, not an annual series. Individual contract deployment dates and the exact extraction date are not recorded in the analytical workbook.

The unit of analysis is a tool and vulnerability category combination. The resulting cross section contains nine tools observed across three categories, producing 27 records. The same tools occur in each category. The annotated vulnerabilities within a category also provide a common denominator for all nine tools. The dataset therefore supports comparisons of reported detection proportions across tools, while the aggregate records do not identify the individual vulnerabilities detected by each tool.

Sample Construction Stage	Tools	Tool and Category Observations	Rule or Outcome
Selected analytical frame	9	27	Nine benchmark tools and three selected vulnerability categories
Missing values in retained fields	0 excluded	0 excluded	No missing values found in the analytical workbook
Duplicate tool and category combinations	0 excluded	0 excluded	No duplicate combinations found
Final descriptive sample	9	27	All selected records retained

The repository is publicly accessible. Its repository materials use the Apache 2.0 licence. The underlying contracts retain their original licences. The workbook does not record a repository commit identifier or the date on which the source table was extracted.

3.2 Data Integration, Cleaning, and Quality Assurance:

The analytical workbook consolidates values from one published results table. The tool name and vulnerability category jointly identify each record. The category label links each detection count to its corresponding annotated vulnerability count. No external dataset, temporal matching, currency conversion, or frequency conversion was required.

Quality checks identified no missing values in the six workbook fields and no duplicate tool and category combinations. Each detection count was within its category denominator, and each recorded detection proportion agreed with the count divided by that denominator. The analysis retained all 27 records. No imputation, interpolation, or outlier removal was applied.

A recorded zero means that the benchmark table reports no matched detections for that tool and category. It does not distinguish a missed vulnerability from a category the tool did not support. Tool design affects which contract properties can be examined, as illustrated by the analysis framework described by Tsankov et al., 2018.

The curated benchmark is a selected collection of annotated vulnerable contracts. It is not a random sample of all Ethereum contracts. Results therefore describe detection within the selected benchmark and should not be interpreted as population estimates for all deployed contracts.

3.3 Variable Measurement, Construction, and Validation:

The independent construct is vulnerability category. It has three observed categories: access control, arithmetic, and reentrancy. These categories distinguish weaknesses in authorization, numerical operations, and contract call sequences. Atzei et al., 2017, provide a taxonomy that supports distinguishing such smart contract vulnerability mechanisms.

The dependent measure is the detection proportion for each tool and category. Equation 1 defines its construction:

$$P_{tc} = \frac{D_{tc}}{N_c}$$

Here, P_{tc} is the detection proportion for tool tand category c, D_{tc} is the number of annotated vulnerabilities matched by the tool; and N_c is the total number of annotated vulnerabilities in that category. The denominators are 19 for access control, 22 for arithmetic, and 8 for reentrancy. A higher proportion indicates that the tool matched a larger share of the annotated vulnerabilities in that category. The measure is a proportion between zero and one, not a percentage.

Variable or Indicator	Analytical Role	Definition and Construction	Source and Identifier	Unit And Frequency	Transformation and Direction	Missing Value Rule	Coverage and Limitation
Analysis tool	Comparison identifier	Name of the tool evaluated	SmartBugs curated results table	Tool; benchmark cross section	Categorical	No missing values found	Nine tools
Vulnerability category	Independent construct	Access control, arithmetic, or reentrancy	SmartBugs curated results table	Category; benchmark cross section	Categorical	No missing values found	Three selected categories
Annotated vulnerabilities	Denominator	Number of annotated vulnerabilities in each category	Curated results table	Vulnerability count; benchmark cross section	No transformation	No missing values found	Common denominator across tools within each category
Matched detections	Outcome component	Annotated vulnerabilities matched by a tool	Curated results table	Vulnerability count; benchmark cross section	Higher values indicate more matched annotations within the same category	No missing values found	Does not identify individual matched vulnerabilities
Detection proportion	Dependent measure	Matched detections divided by annotated vulnerabilities	Calculated from curated results table	Proportion; benchmark cross section	Range from zero to one; higher values indicate a larger detected share	No missing values found	Describes matched detection within the curated benchmark

The proportion measures coverage of annotated vulnerabilities. It does not measure the proportion of a tool's warnings that are correct. Feist et al., 2019, show that vulnerability detection and false positives are separate aspects of tool evaluation. The aggregate workbook does not provide the information needed to calculate precision.

Vulnerability Category	Valid Observations	Mean	Standard Deviation	Median	Minimum	Maximum
Access control	9	0.082	0.102	0.000	0.000	0.211
Arithmetic	9	0.217	0.279	0.045	0.000	0.682
Reentrancy	9	0.472	0.311	0.625	0.000	0.875
All selected records	27	0.257	0.290	0.182	0.000	0.875

The descriptive statistics summarize tool and category records. The overall mean gives equal weight to each record and should not be interpreted as the proportion of all unique benchmark vulnerabilities detected. The category distributions describe the available benchmark results without establishing why the proportions differ.

3.4 Analytical Framework, Identification, and Reproducibility:

The analysis is descriptive. For each vulnerability category, it compares the reported detection proportion across the nine tools. The relevant variation is the difference in matched detection counts among tools evaluated against the same category denominator. Category means and ranges provide concise summaries of these comparisons.

The data do not support a causal estimate of the effect of vulnerability category or tool design on detection. The categories contain different annotated examples, and the tools may differ in their supported properties. The aggregate table does not identify which vulnerabilities were detected by multiple tools, so the

observations cannot establish agreement or disagreement for individual vulnerabilities. Durieux et al., 2020, provide the benchmark context for interpreting these comparisons.

The analytical workbook preserves the 27 source counts, category denominators, and formulas used to calculate the proportions. The tool and category fields allow each record to be checked against the curated results table. The available version supports reproduction of the descriptive summaries, subject to the absence of a recorded source commit and extraction date.

4. Methods:

4.1 Research Design and Analytical Scope:

This analysis uses a descriptive secondary design to compare automated detection results across nine tools and three categories of annotated Ethereum smart contract vulnerabilities. The unit of analysis is a tool and vulnerability category combination. Each research question concerns variation in the reported detection proportion across tools within one category.

The comparison uses the common category denominator supplied by the curated benchmark. It describes detection in the selected cases without treating vulnerability category as an assigned treatment. The original benchmark was developed and evaluated by Durieux et al., 2020; the present analysis uses its published aggregate results rather than rerunning the tools.

Differences in detection proportions may reflect tool capabilities, benchmark composition, or the way vulnerabilities were annotated and matched. The aggregate data do not separate these explanations. Accordingly, the comparisons are not interpreted as causal effects or as estimates of performance across all deployed contracts.

4.2 Measurement and Descriptive Specification:

The sample construction table and variable dictionary in Section 3 define the selected records and their measures. The outcome for each tool and category is the detection proportion defined in Equation 1: matched detections divided by the annotated vulnerability count for that category. The analysis retains the underlying counts so that the reported proportions can be checked against their denominators.

For each category, the analysis summarizes the nine tool proportions using their arithmetic mean, median, sample standard deviation, minimum, and maximum. The category mean is calculated as:

$$\bar{P}_c = \frac{1}{9}\sum_{t=1}^{9} P_{tc}$$

Here, $\bar{P}_c$ is the mean detection proportion for category c, P_{tc} is the proportion for tool t in that category, and nine is the number of tools. Each tool receives equal weight. The mean describes the nine reported tool results; it is not the proportion of unique vulnerabilities detected by at least one tool.

The analysis also compares matched counts and percentages across tools within each category. Percentages are obtained by multiplying the proportions by 100 and are rounded to one decimal place for presentation. Ren et al., 2021, show why common evaluation settings and clearly defined measures matter when comparing smart contract testing tools.

4.3 Sample Alignment and Data Processing:

The analytical sample contains 27 records, comprising all nine tools across the three selected categories. The data are a benchmark cross section, not a time series or panel. The source benchmark contains annotated vulnerable contracts, but the workbook records category level counts rather than individual contract outcomes.

The processing sequence uses the tool name and category as the joint record identifier. The analysis checks each record for missing values, duplicate identifiers, positive denominators, detection counts within their denominators, and agreement between the recorded proportion and the count divided by the denominator. These checks found no missing or duplicate records in the analytical workbook and no inconsistent proportions. All 27 records were retained; no imputation or removal of extreme values was applied.

The category denominators differ, so an equal change in the number of matched vulnerabilities produces a different percentage point change across categories. Comparisons across categories therefore describe results for different sets of annotations. A zero matched count remains a reported zero; the aggregate table does not distinguish an unsupported category from an undetected vulnerability.

4.4 Interpretation, Validation, and Reproducibility:

The research questions are answered through observed tool comparisons and category summaries. No regression model, mediation analysis, statistical significance test, or causal estimator is applied. The same annotated vulnerabilities are used across tools within a category, while the workbook does not identify individual matches. Consequently, the aggregate records do not supply the paired detection information needed to assess overlap or conduct a paired comparison of individual vulnerabilities.

The checks establish internal consistency of the workbook values and calculations. They do not independently verify the original annotations or measure false positives. Ghaleb and Pattabiraman, 2020,

demonstrate that an evaluation of detection tools can require evidence on both missed vulnerabilities and incorrect warnings. The present outcome addresses matched detection only.

The descriptive calculations were checked using Python version 3.12.14 and openpyxl version 3.1.5. Replication requires the analytical workbook or the curated results table in the SmartBugs repository, the category selection, and the calculations described above. The workbook records the source and formulas but does not specify a repository commit or extraction date. The method therefore supports reproduction of the reported descriptive comparison, subject to those provenance limits.

5. Findings:

This analysis compares detection proportions across nine automated tools for three categories of annotated Ethereum smart contract vulnerabilities. The findings address each research question using the same benchmark results. They show how detection varies across tools and categories within the observed sample.

5.1 Sample and Detection Measures:

The analytical sample contains 27 tool and category observations: nine tools assessed against access control, arithmetic, and reentrancy vulnerabilities. All records contain complete detection counts and denominators. The underlying benchmark contains 69 annotated vulnerable contracts, although the workbook reports aggregate results rather than individual contract observations. The benchmark results were originally published by Durieux et al., 2020.

The outcome is the proportion of annotated vulnerabilities matched by each tool. The denominators are 19 access control vulnerabilities, 22 arithmetic vulnerabilities, and 8 reentrancy vulnerabilities. Table 1 reports the matched count and detection percentage for each tool. A zero indicates no matched detections in the source table. It does not establish whether the tool lacked support for the category or failed to detect supported cases.

Table 1: Detection of annotated vulnerabilities by tool and category

Analysis Tool	Access Control, Count and Percentage	Arithmetic, Count and Percentage	Reentrancy, Count and Percentage
Honeybadger	0 of 19; 0.0%	0 of 22; 0.0%	0 of 8; 0.0%
Maian	0 of 19; 0.0%	0 of 22; 0.0%	0 of 8; 0.0%
Manticore	4 of 19; 21.1%	4 of 22; 18.2%	2 of 8; 25.0%
Mythril	4 of 19; 21.1%	15 of 22; 68.2%	5 of 8; 62.5%
Osiris	0 of 19; 0.0%	11 of 22; 50.0%	5 of 8; 62.5%
Oyente	0 of 19; 0.0%	12 of 22; 54.5%	5 of 8; 62.5%
Securify	0 of 19; 0.0%	0 of 22; 0.0%	5 of 8; 62.5%
Slither	4 of 19; 21.1%	0 of 22; 0.0%	7 of 8; 87.5%
Smartcheck	2 of 19; 10.5%	1 of 22; 4.5%	5 of 8; 62.5%

Note. Percentages are calculated from the counts in the attached analytical workbook and rounded to one decimal place. The benchmark results are reported in the SmartBugs repository.

5.2 Access Control Detection:

For RQ1, detection of access control vulnerabilities ranged from 0.0% to 21.1%. Manticore, Mythril, and Slither each matched 4 of the 19 annotated vulnerabilities. Smartcheck matched 2, while the remaining five tools recorded no matches. The mean detection proportion across the nine tools was 8.2%, and the median was 0.0%.

The highest reported count leaves 15 of the 19 annotations unmatched by each leading tool. This result describes coverage in the selected benchmark. It does not establish that the tools identified the same four vulnerabilities. Tsankov et al., 2018, describe security analysis as checking contract behavior against specified properties, a distinction relevant when interpreting category level detection results.

5.3 Arithmetic Detection:

For RQ2, arithmetic detection ranged from 0.0% to 68.2%. Mythril matched 15 of 22 annotated vulnerabilities, followed by Oyente with 12 and Osiris with 11. Manticore matched 4, Smartcheck matched 1, and four tools recorded no matches. The mean across tools was 21.7%, while the median was 4.5%.

The 68.2 percentage point gap between the highest and lowest reported proportions shows substantial variation within this category. Ferreira Torres et al., 2018, developed Osiris to detect integer weaknesses through symbolic execution and data tracing. Its result in this benchmark shows detection of half the annotated arithmetic cases; the aggregate data do not establish why its count differs from those of other tools.

5.4 Reentrancy Detection:

For RQ3, reentrancy detection ranged from 0.0% to 87.5%. Slither matched 7 of 8 annotated vulnerabilities. Mythril, Osiris, Oyente, Securify, and Smartcheck each matched 5. Manticore matched 2, while Honeybadger and Maian recorded no matches. The mean across tools was 47.2%, and the median was 62.5%.

Slither's result indicates the highest observed reentrancy coverage in this benchmark. Feist et al., 2019, describe Slither as a framework that applies program analysis techniques to Solidity contracts. The benchmark count alone does not identify the specific mechanism responsible for its observed performance. Because the category contains eight annotations, a difference of one matched vulnerability changes the reported proportion by 12.5 percentage points.

5.5 Cross Category Interpretation:

The mean detection proportion across tools was highest for reentrancy at 47.2%, followed by arithmetic at 21.7% and access control at 8.2%. The same ranking appears in the category medians. These summaries answer the three research questions for the selected benchmark, but they do not establish that one vulnerability category is inherently easier to detect. The categories contain different annotated cases and denominators, and the tools may cover different properties.

The results also show that no single tool had the highest reported proportion in every category. Mythril shared the highest access control proportion and led arithmetic detection, while Slither led reentrancy detection. Luu et al., 2016, illustrate that automated analysis tools can target specific classes of smart contract behavior. The present comparison does not measure whether combining tools would improve coverage because the workbook does not identify overlaps among their detections.

The findings are descriptive comparisons of previously published benchmark results. They do not measure false positive rates, statistical uncertainty, or performance across the full population of Ethereum contracts. They also do not independently validate the tools or establish a causal relationship between vulnerability category and detection.

6. Discussion:

A tool that detects one class of smart contract vulnerability well may offer less coverage in another. Table 1 shows this asymmetry: Mythril had the highest reported arithmetic detection proportion, while Slither led reentrancy detection and shared the highest access control proportion. This pattern gives the conceptual framework a practical interpretation: vulnerability category matters when comparing tool results, and an overall ranking can conceal differences relevant to an audit. The analysis provides a focused comparison of three categories in an existing benchmark. Because the underlying results were previously published, it should be presented as a secondary analysis rather than as evidence of a previously undocumented discovery. Ren et al., 2021, show that evaluation settings can materially affect conclusions about smart contract testing tools, reinforcing the need to interpret each comparison within its benchmark conditions.

The arithmetic and reentrancy results suggest that detection should be evaluated against the behavior a tool must recognize. Arithmetic weaknesses concern numerical operations, whereas reentrancy involves sequences of calls and state changes. Equation 1 measures the share of annotations detected within each category, but it does not identify which features of a tool produced a match. Chen et al., 2019, found that limitations in identifying contract control flows affected static analysis and that execution traces could complement analysis in their evaluation. That finding offers a plausible explanation for variation across tools, but the present aggregate data do not test whether control flow identification caused the observed differences.

The lower reported coverage of access control annotations is relevant to the interpretation of security assessments. In Table 1, the highest individual tool proportion for this category was 21.1%, while five tools recorded no matched detections. This pattern indicates limited coverage of the selected annotations by individual tools; it does not demonstrate that the tools would perform similarly on every access control weakness. It also does not show whether their detections overlap. Ghaleb and Pattabiraman, 2020, found missed vulnerabilities and false positives when evaluating tools with injected bugs, indicating that both missed detections and incorrect warnings matter when judging an analysis tool. The present detection proportions address the first issue only for the selected annotations.

The difference between benchmark detection and operational protection also shapes the international relevance of the findings. Ethereum contracts can be used across jurisdictions, so the observed category differences are relevant to auditors assessing contracts for users in different markets. The dataset does not identify the location of contract users or provide a basis for comparing advanced and emerging economies. Rodler et al., 2019, examine a different security setting by evaluating runtime protection against reentrancy in deployed contracts. Their work shows that detection during code analysis and protection during execution address related but distinct stages of security. The present results do not measure runtime protection or establish how it would change the benchmark outcomes.

For audit planning, the results support examining tool coverage separately for each vulnerability category rather than relying on a single overall detection figure. The recorded access control results also support retaining focused human review of authorization logic alongside automated analysis. Before choosing a combination of tools, auditors and researchers need case level evidence showing which annotations each tool detects, as well as evidence on false positives. Such data would allow future work to evaluate whether tools detect distinct cases, whether their combination improves coverage, and whether the observed category patterns

persist in other benchmarks. Those questions extend the present descriptive comparison without assuming that a particular tool combination is effective.

7. Conclusion and Implications:

The security of smart contracts used across borders depends on recognizing that automated tools offer uneven coverage of potential weaknesses. This analysis shows that detection performance varies across the examined categories and that no single tool leads in all three. Its contribution is a focused secondary comparison of published benchmark results, bringing category differences into view for decisions about security review. The findings qualify the use of a single overall tool ranking: effective assessment requires attention to the type of behavior being examined. Developers and auditors can use this insight to select checks suited to the risks in a contract and retain focused human review where automated coverage is limited. Organizations responsible for security assurance can also require reports that distinguish detected vulnerabilities from unsupported or unexamined categories. Clearer reporting can help users interpret the scope of an audit, although the evidence does not establish the effectiveness of any particular audit process.

The analysis is limited to aggregate results from a curated benchmark. It cannot identify overlapping detections, establish why tools differ, or measure false positive rates. Future research can use case level annotations to evaluate whether tools detect complementary weaknesses, then assess both missed vulnerabilities and incorrect warnings across independent contract collections. Repeated evaluations with different tool versions and contract environments could determine whether the observed patterns remain useful beyond this benchmark.

References:

1. Atzei, N., Bartoletti, M., & Cimoli, T. 2017. A survey of attacks on Ethereum smart contracts. In Principles of Security and Trust, Lecture Notes in Computer Science, volume 10204, pages 164-186. Springer. https://doi.org/10.1007/978-3-662-54455-6_8
2. Chen, T., Li, Z., Zhang, Y., Luo, X., Wang, T., Hu, T., Xiao, X., Wang, D., Huang, J., & Zhang, X. 2019. A large scale empirical study on control flow identification of smart contracts. In Proceedings of the 13th ACM and IEEE International Symposium on Empirical Software Engineering and Measurement, article 8870156. IEEE. https://doi.org/10.1109/ESEM.2019.8870156
3. Durieux, T., Ferreira, J. F., Abreu, R., & Cruz, P. 2020. Empirical review of automated analysis tools on 47,587 Ethereum smart contracts. In Proceedings of the ACM and IEEE 42nd International Conference on Software Engineering, pages 530-541. IEEE. https://doi.org/10.1145/3377811.3380364
4. Feist, J., Grieco, G., & Groce, A. 2019. Slither: A static analysis framework for smart contracts. In Proceedings of the IEEE and ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, pages 8-15. IEEE. https://doi.org/10.1109/WETSEB.2019.00008
5. Ferreira Torres, C., Schütte, J., & State, R. 2018. Osiris: Hunting for integer bugs in Ethereum smart contracts. In Proceedings of the 34th Annual Computer Security Applications Conference. Association for Computing Machinery. https://doi.org/10.1145/3274694.3274737
6. Ghaleb, A., &Pattabiraman, K. 2020. How effective are smart contract analysis tools? Evaluating smart contract static analysis tools using bug injection. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. Association for Computing Machinery. https://doi.org/10.1145/3395363.3397385
7. Luu, L., Chu, D. H., Olickel, H., Saxena, P., & Hobor, A. 2016. Making smart contracts smarter. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pages 254-269. Association for Computing Machinery. https://doi.org/10.1145/2976749.2978309
8. Ren, M., Yin, Z., Ma, F., Xu, Z., Jiang, Y., Sun, C., Li, H., & Cai, Y. 2021. Empirical evaluation of smart contract testing: What is the best choice? In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, pages 566-579. Association for Computing Machinery. https://doi.org/10.1145/3460319.3464837
9. Rodler, M., Li, W., Karame, G. O., & Davi, L. 2019. Sereum: Protecting existing smart contracts against reentrancy attacks. In Proceedings of the Network and Distributed System Security Symposium. Internet Society. https://doi.org/10.14722/ndss.2019.23413
10. SmartBugs. 2020. Vulnerability analysis of smart contracts using SmartBugs. Data repository. https://github.com/smartbugs/smartbugs-results
11. Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Bünzli, F., & Vechev, M. 2018. Securify: Practical security analysis of smart contracts. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pages 67-82. Association for Computing Machinery. https://doi.org/10.1145/3243734.3243780