Evaluation and Classification of Smart Contract Analysis Tools in Blockchain

Table of Contents

List of Figures

List of Tables

Glossary

Bytecode: Bytecode is computer object code that an interpreter converts into binary machine code so it can be read by a computer's hardware processor.

Code instrumentation framework: Instrumentation refers to the measure of a product's performance, in order to diagnose errors and to write trace information.

Concolic testing: Concolic testing is a hybrid software verification technique that performs symbolic execution along a concrete execution (testing on particular inputs) path.

DAO attack: The decentralized autonomous organization was hacked due to vulnerabilities in its code base in 2016.

Dynamic analysis: Dynamic analysis is the testing and evaluation of a program by executing data in real-time. The objective is to find errors in a program while it is running.

False Negative: A test result which wrongly indicates that a particular condition or attribute is absent.

False Positive: A test result which wrongly indicates that a particular condition or attribute is present.

Fuzzing: Fuzzing is a faster and cheaper method for detecting some types of vulnerabilities than exhaustive, line-by-line analysis of application code.

Gasless send: Gasless refers to a model in which users do not bear the cost of transaction fees. A third-party can send another user's transactions and pay themselves for the gas cost.

Integer overflow: Integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of digits.

Lexical analysis: It gathers modified source code that is written in the form of sentences from the language preprocessor. It is responsible for breaking these syntaxes into a series of tokens, by removing whitespace in the source code.

Mutated contracts: The contract does allow for any modifications and/or reversal.

P value: The p value tells you how likely it is that your data could have occurred under the null hypothesis.

Receiver Operating Characteristic (ROC): A plot of test sensitivity as the y coordinate versus its 1-specificity or false positive rate (FPR) as the x coordinate. It is used to assess the overall diagnostic performance of a test and to compare the performance of two or more diagnostic tests.

Re-entrancy: A reusable routine that multiple programs can invoke, interrupt, and reinvoke simultaneously.

Runtime verification: System analysis and execution approach based on extracting information from a running system and using it to detect and possibly react to observed behaviors satisfying or violating certain properties.

Smart Contracts: Smart contracts are programs stored on a blockchain that run when predetermined conditions are met

Static analysis: Automated analysis of source code without executing the application.

Symbolic Execution: A way of executing a program abstractly, so that one abstract execution covers multiple possible inputs of the program that share a particular execution path through the code.

Syntax analysis: It checks the syntactical structure of the given input, i.e. whether the given input is in the correct syntax.

Syntax tree: It is a tree representation of the abstract syntactic structure of text (often source code) written in a formal language.

Timestamp dependency: Timestamp dependence vulnerability is when a smart contract utilizes the block. timestamp function when performing critical logic in a smart contract.

True negative: It is an outcome where the model correctly predicts the negative class.

True positive: It is an outcome where the model correctly predicts the positive class.

Unhandled exception: An unhandled exception is an error in a computer program or application when the code has no appropriate handling exceptions.

Xpath patterns: The XML Path Language (XPath) is used to uniquely identify or address parts of an XML document. An XPath expression can be used to search through an XML document, and extract information from any part of the document, such as an element or attribute. XPath uses path expressions to select nodes or node-sets in an XML document

1 Introduction

Smart contracts are self-enforcing agreements, a piece of software which resides and runs over blockchains. They are used for secure and transparent management of the authentication, connection, and transaction processes in decentralized blockchain environments, mostly Internet of Things, which is a niche area of research and practice. However, writing efficient and effective smart contracts can be tremendously challenging due to absence of any standard body and tools, and lack of any standard benchmarking.

2 Project Vision

This research looks forward to analyzing the smart contracts to initially benchmark some popular state-of-the-art existing tools. After benchmarking we shall try to improve a selected tool to achieve greater yield by reducing the rate of false positives.

2.1 Problem Domain

Vulnerabilities targeting smart contracts have been on the rise, which have led to financial loss and erosion of trust. There exist a number of static analysis tools with a high number of false positives in smart contracts. However, despite the numerous bug-finding tools, there is no systematic approach to evaluate the proposed tools and gauge their effectiveness. This paper proposes to develop a standard benchmark based on state-of-the-art popular tools, and to uses symbolic execution to address false positives.

2.2 Research Problem Statement

Executing, verifying, and enforcing reliable transactions on blockchains is done using automated agreements. These agreements, known as smart contracts, are analyzed by different tools. These tools give us different false positives which are used to analyze smart contacts. A fundamental problem with analyzing these smart contracts is ensuring their correctness and reliability.

To date, the state-of-the-art analysis techniques in the literature rely on static or dynamic analysis with massively high rate of false positives or lack of support for vulnerabilities like out of gas and unchecked send.

2.3 Business Opportunity

This research project will be beneficial to blockchain researchers and developers who have a stake in smart contracts and blockchain technology. It will also help them reduce the number of false positives in smart contracts encountered using analysis tools.

2.4 Objectives

There are two main objectives of this research project.

- Improve an existing tool by reducing the number of false positives.
- Make a machine learning model which will help us classify any new tool.

2.5 Scope

Our research project is based on smart contracts in blockchain technology. We will be analyzing different tools and their outputs. We will create a well-defined benchmark to analyze the tools and see how many false positives we can filter in the output. However, we will not be validating any smart contracts. The analysis will lead us to focus our research on a specific tool which will lead us to a better result by reducing rate of false positives. We may also use symbolic execution or static analysis to reduce the number of false positives and integrate a machine learning model in our research which will classify new tools according to the type of problem they may have. All of our work will be done in command line.

2.6 Constraints

Our constraints are limited to the number of analysis tools and smart contracts we have available online. We will choose the most common tools. Another constraint is the amount of time it takes for tools to analyze a smart contract.

2.7 Stakeholder and User Descriptions

The following tells us about the stakeholders involved.

2.7.1 Market Demographics

The market demographics consists of researchers and developers who are interested in blockchain technology specifically smart contracts. They will be interested in our project since it offers research to reduce the number of false positives. Apart from them, the regulator and blockchain user will also be interested in this project.

2.7.2 Stakeholder Summary

Name	**Description**	**Responsibilities**
Project researchers	It includes those who are researching and developing this project	1. Should research about different analysis tools 2. Should analyze the different tools selected

		3. Improve an existing tool to reduce false positives
End-User	End user involves blockchain researchers and developers	1. Should study and evaluate the project independently
Supervisor of project	Supervisor will be involved and giving directions for this research project	1. Gives the team directions 2. Ensures quality work 3. Monitors the project's progress 4. Ensures that the research is headed in the right direction

Table 1: Stakeholder Summary

2.7.3 Stakeholder Profiles

Representatives	Supervisor: Dr. Affan Rauf
Description	Dr. Affan is involved in supervising activities of research process
Type	He is a technical stakeholder. He has expertise in domains in which the research is being conducted
Responsibilities	1. Gives direction to research team 2. Ensures quality work 3. Monitors the project's progress 4. Facilitate research team to complete the project in time 5. Ensures that research is headed in the right direction
Success Criteria	Completion of research objectives which are being committed
Involvement	1. Reviews research 2. Guides the research team
Comments/Issues	None

Table 2: Supervisor of the Project

Representatives	Mr. Muhammad Ali Amer Mr. Muhammad Ali Hassan Ahmad
Description	They are involved in the research process

Type	They are technical stakeholders
Responsibilities	1. Should research about different analysis tools 2. Should analyze the different tools selected 3. Improve an existing tool to reduce false positives 4. Integrate a machine learning model
Success Criteria	Completion of research objectives which are being committed
Involvement	1. Research 2. Tool selection 3. Tool improvement 4. Machine learning model
Comments/Issues	None

Table 3: Research Team of the Project

2.7.4 User Environment

Any user who know UNIX operating system and has experience in it should be able to use the different analysis tools.

3 System Requirement Specification

The following tells us about the system requirements.

3.1 System Features

- Analyze tools and create error matrix along with benchmark.
- Improve an analysis tool which will give reduced false positives
- Integrate a machine learning model which will classify new tools according to the type of problem it may have

3.2 Functional Requirements

The functional requirements are listed below.

- We will categorize the analysis tools according to number of false positives.
- We will analyze and compare the different number of analysis tools based on the amount of false positives they give.
- We will take a specific analysis tool and reduce the false positive errors it gives.
- We will integrate a machine learning model.

3.3 Non-Functional Requirements

The non-functional requirements are listed below.

- Reduce the number of false positives in smart contract analysis tools.
- The machine learning algorithm will classify any new tool according to the type of problem it may have.

4 Literature Review

Prior to the first paper, “How Effective are Smart Contract Analysis Tools? Evaluating Smart Contract Static Analysis Tools Using Bug Injection [1]” by Asem Ghaleb, there was no systematic approach to evaluate the proposed tools and gauge their effectiveness. All the tools report numerous false positives. Static analysis tools have only been evaluated either by their developers on custom datasets and inputs, with a limited number of bugs. Many of the defects can be detected by static analysis tools in theory, but are not detected due to limitations of the tools.

The author in this paper [1] proposes a systematic approach to evaluate the proposed analysis tools and their effectiveness. To do this, the author uses a tool called SolidiFI. SolidiFI injects bugs into all potential locations in a smart contract to introduce targeted security vulnerabilities. SolidiFI then checks the generated buggy contract using the static analysis tools, and identifies the bugs that the tools are unable to detect along with identifying the bugs reported as false-positives.

The author [1] has a dataset of 50 smart contracts in which they inject 9369 distinct bugs. There were 7 main bugs which were injected namely, 1. Re-Entrancy 2. Timestamp dependency 3. unchecked send 4. unhandled exceptions 5. TOD 6. Integer overflow/underflow 7. use of tx.origin. SolidiFI automates the process of injecting bugs. Then these were evaluated by 6 smart contract analysis tools which were Oyente, Securify, Mythril, SmartCheck, Manticore and Slither. The experiment injects the bugs into the smart contract and then evaluates them using the analysis tools. Then the paper evaluates whether the tool correctly identified all the injected bugs.

The evaluation results show several cases of bugs that were not detected by the evaluated tools even though those undetected bugs are within the detection scope of tools. SolidiFI thus identifies important gaps in current static analysis tools for smart contracts, and provides a reproducible set of tests for developers of future static analysis tools.

This paper [1] is mainly focused to false negatives. The author limit themselves to only static analysis tools which are evaluated by a tool SolidiFI. Many of the detected bugs were not caught by the analysis tools

such as Mythril failed to detect the largest set of bugs in the experiments. Each analysis tool was unable to identify different bugs which were present in the contract. This experiment allows smart contract developers to understand the limitations of existing static analysis tools with respect to detecting security bugs.

This research article [1] focuses on false negatives by injecting bugs in the smart contract. This provides a basic understanding to us on what tools detect what bugs and their performance. This will help us when benchmarking to know which tools can detect and miss what vulnerabilities in smart contract.

Security bug	Injected bugs	Oyente	Securify	Mythril	SmartCheck	Manticore	Slither
Re-entrancy	1343	1008 (844)	232 (232)	1085 (805)	1343 (106)	1250 (1108)	✓
Timestamp dep	1381	1381 (886)	NA	810 (810)	902 (341)	NA	537 (1)
Unchecked-send	1266	NA	499 (449)	389 (389)	NA	NA	NA
Unhandled exp	1374	1052 (918)	673 (571)	756 (756)	1325 (1170)	NA	457 (128)
TOD	1336	1199 (1199)	263 (263)	NA	NA	NA	NA
Integer overflow	1333	898 (898)	NA	1069 (932)	1072 (1072)	1196 (1127)	NA
tx.origin	1336	NA	NA	445 (445)	1239 (1120)	NA	✓

Figure 1: Number of False Negatives identified for each tool [1]

The second paper, “Empirical Vulnerability Analysis of Automated Smart Contracts Security Testing on Blockchains [2]“ by Reza M. Parizi mainly tries to answer two different questions relating to smart contract analysis tools. The first is, ‘How effective are the automated smart contract security testing tools, in terms of vulnerability detection ability? And what is the most effective tool?’ and the second is ‘What are the accuracy scores obtained by the automated testing security tools in detecting true vulnerabilities?’

There are 4 main tools that this paper [2] uses namely, Oyente, Mythril, Securify, and SmartCheck. The paper [2] uses 10 smart contracts to experiment on. The experiment selects a new tool randomly and a new smart contract randomly. It then applies the tool on the smart contract and collects the results. One by one all tools are applied on all contracts. The visualization technique for performance of tool was Receiver Operating Characteristic (ROC). It showed the tool accuracy and the resulting curves.

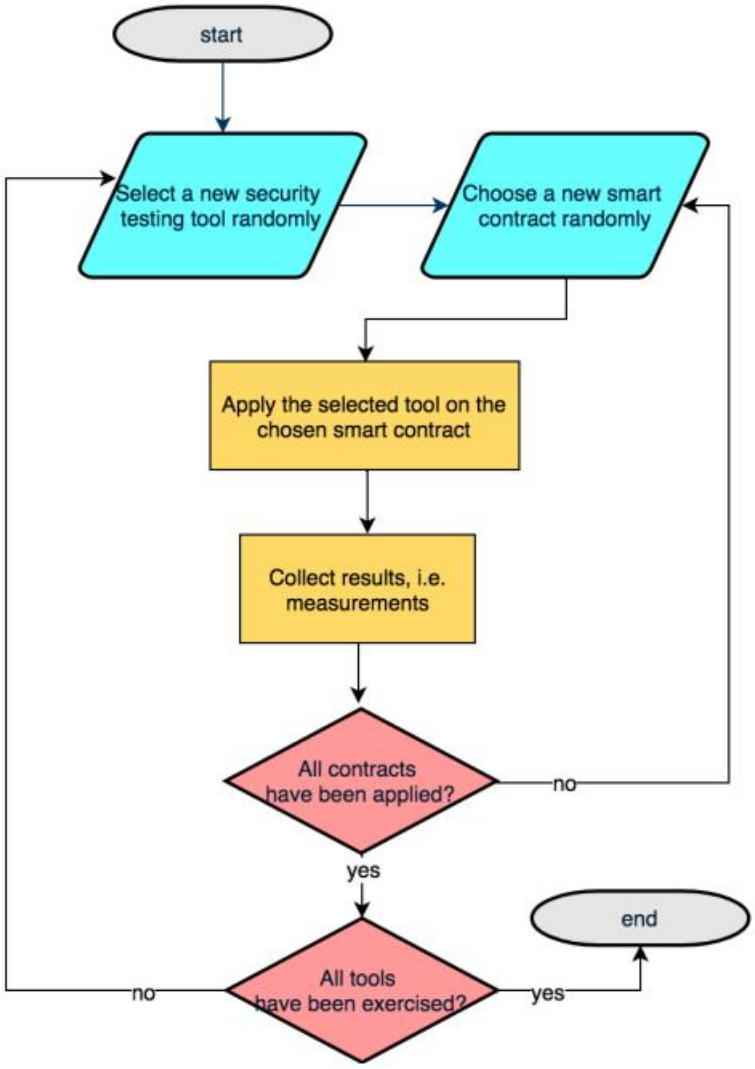

Figure 2: General Execution Process implemented in the paper [2]

The paper [2] showed that SmartCheck tool is statistically more effective than the other automated security testing tools at 95% significance level ($p < 0.05$). Concerning the accuracy, Mythril was found to be significantly ($p < 0.05$) accurate with issuing the lowest number of false alarms among peer tools. These results could imply that SmartCheck could currently be the most effective static security testing tool for Solidity smart contracts on the Ethereum blockchain but perhaps less accurate than Mythril.

This paper [2] runs 4 tools on smart contracts and evaluates its performance and accuracy using different techniques. SmartCheck tool shows the highest performance in the ROC plot and Oyente tool performed the worst according to this research paper. On evaluating accuracy, Mythril tool showed the highest accuracy score however Oyente tool again performed the worst in accuracy.

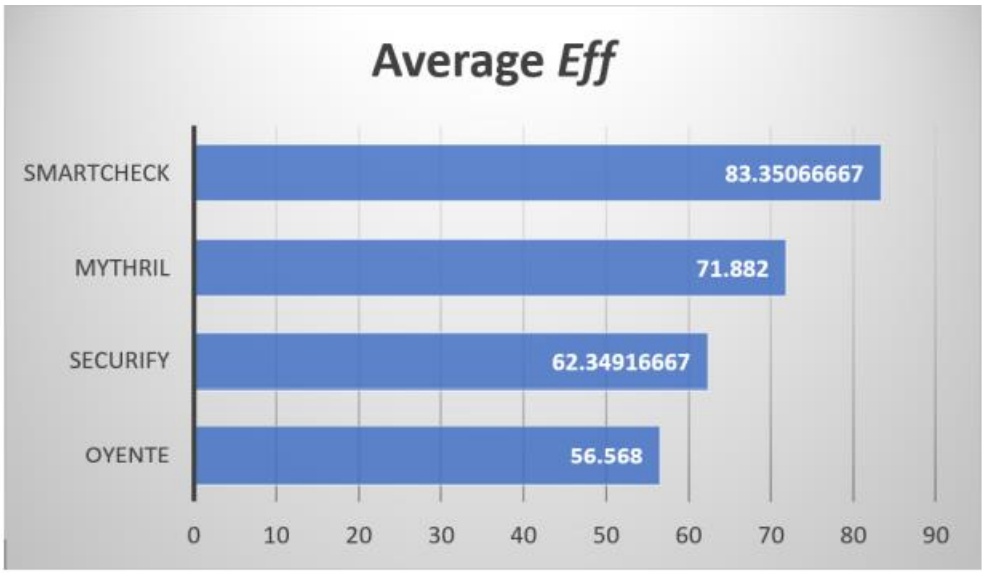

Figure 3: Final Effectiveness Rate of the tools [2]

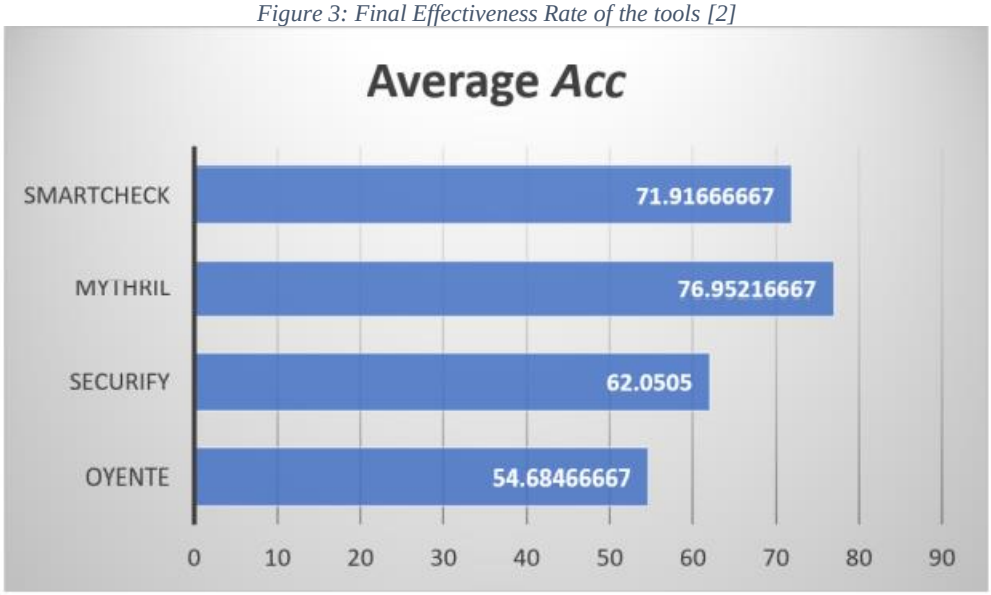

Figure 4 Final Accuracy Rate of the tools [2]

This paper [2] tells us about the different performance and accuracy scores of analysis tools. This can help us in evaluating which tool we should use for analysis techniques and which tool to avoid. We are already using Mythril tool and will take help from this paper to select next tool based on effectiveness and accuracy.

The next research paper "Security Vulnerabilities in Ethereum Smart Contracts [3]" by Ardit Dika examines the different audited and vulnerable smart contracts and uses different analysis tools to gauge their effectiveness, accuracy and consistency. This paper [3] analyses both bytecode and solidity contracts. It focuses on both false-negatives and false-positives.

The paper [3] uses 21 audited smart contracts and 24 vulnerable smart contracts. 4 tools were used to analyse the smart contracts namely, Oyente, Securify, Remix and SmartCheck. The audited smart contracts are used for false positives and vulnerable smart contracts are used for false negatives. The paper [3] also outlines vulnerabilities and classifies them based on their architectural and severity level.

The paper [3] classifies the tools based on the methodology they use, the user interface, and the analysis they are able to execute, which allows them to build a 'state of the art' security tools on Ethereum. They also construct a matrix of security tools and the vulnerabilities they cover in order to identify gaps and absent vulnerability check.

The paper [3] analyses different smart contracts and analyzes them according to the false positives and false negatives. It then creates a vulnerability matrix to find which tool accurately catches what vulnerability. This is beneficial as this gauges the overall effectiveness and accuracy of the tool. According to this paper [3], SmartCheck found out the most vulnerabilities.

This paper [3] tells us a lot about the different analysis tools and the vulnerabilities they catch. It also provides us with resources from where to collect both audited and vulnerable smart contracts.

Security Tool	ReEntrancy	Timestamp dependency	TOD	Mishandled exceptions	Immutable Bugs	tx.origin usage	Gas costly patterns	Blockhash usage
Oyente	✓	✓	✓	✓	✓	×	×	×
Remix	✓	✓		✓	×	✓	✓	✓
F*	✓	×	×	✓	×	×	×	×
Gasper	×	×	×	×	×	×	✓	×
Securify	✓	×	✓		×	✓	×	×
S. Analysis	×	×		✓	×	×	×	×
SmartCheck	✓	✓	✓	✓	✓	✓	✓	×
Imandra	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Mythril	✓	×	×	✓	✓	✓	×	×

Figure 5: Vulnerability Matrix for finding out which tool detects which vulnerability [3]

This paper "Security Vulnerabilities in Ethereum Smart Contracts [4]" by Alexander Mense summarizes known vulnerabilities in smart contracts found by literature research. It compares currently available contract analysis tools for their capabilities to identify and detect vulnerabilities in smart contracts based on taxonomy for vulnerabilities. This paper [4] also shows an example for best practices to avoid vulnerabilities in smart contracts through The DOA attack.

This paper [4] lists down the different vulnerabilities present in the smart contracts and classifies them according to the severity level. These vulnerabilities include but not limited to, gasless send, external calls and mishandled exceptions. The paper [4] then uses analysis tools namely, Oyente, Securify, Remix, SmartCheck, F*, Mythril, and Gasper. The aim of The DOA attack is to create an organization with decentralized control.

Re-entrancy ranks the highest among all the vulnerabilities with 6 out of 7 tools detecting it. The results show the different analysis tools detecting different vulnerabilities.

This paper [4] lists down vulnerabilities and analyzes smart contracts using different analysis tools. The paper**Error! Bookmark not defined.** tells us how to avoid re-entrancy issue by explaining the code and suggesting that internal work should be finished first and then call the external contract.

This paper [4] will help us know how to improve smart contracts and different vulnerabilities detected by the tool. The paper also tells us how some tools focus on sever vulnerabilities only and some have a more comprehensive vulnerability detection capability. The following figure provides the results.

	Vulnerability	Oyente	Securify	Remix	SmartCheck	F*	Mythril	Gasper	Sum
Solidity	Gas costly patterns	✓	×	✓	✓	×	×	✓	4
	Call to the unknown	×	×	×	×	×	✓	×	1
	Gasless send	×	×	×	×	×	×	×	0
	Mishandled exceptions	✓	✓	✓	✓	✓	×	×	5
	Type casts	×	×	×	×	×	✓	×	1
	Re-entrancy	✓	✓	✓	✓	✓	✓	×	6
	Unchecked math	×	×	×	✓	×	✓	×	2
	Exposed functions	×	✓	✓	×	×	✓	×	3
	'tx.origin' usage	✓	✓	✓	✓	×	✓	×	5
	'blockhash' usage	✓	✓	×	✓	×	×	×	3
	Denial of Service (DoS)	×	×	×	✓	×	×	×	1
	'send' instead of 'transfer'	×	×	×	✓	×	×	×	1
	Style violation	×	×	×	✓	×	×	×	1
	Redundant fallback function	×	×	×	✓	×	×	×	1
EVM	Immutable bugs	×	✓	×	×	×	×	×	1
	Ether lost in transfer	×	✓	×	✓	×	×	×	2
Blockchain	Unpredictable state	×	×	×	×	×	×	×	0
	Generating randomness	×	×	×	×	×	✓	×	1
	Timestamp dependency	✓	×	✓	✓	×	✓	×	4
	Lack of transactional privacy	×	×	×	×	×	×	×	0
	Transaction-ordering dependency	✓	✓	×	×	×	✓	×	3
	Untrustworthy data feeds	×	×	×	×	×	×	×	4
	Sum	7	8	6	12	2	9	1	

Figure 6: Vulnerability Detection Ability of Tools [4]

In this research paper, “Analysis of Blockchain Smart Contracts: Techniques and Insights [5]” by S.Kim, they collected 391 papers and extracted 67 relevant to the smart contract analysis. These papers were classified into 3 different topics mainly, static analysis for vulnerability detection, static analysis for program correctness and dynamic analysis, which were then explored with further classifications.

The extracted papers either statically or dynamically analyzed smart contracts. In static analysis for vulnerability detection, symbolic execution, which simulates concrete executions with variable symbolic value inputs. Concolic testing is the most dominant technique for vulnerability detection in EVM bytecode.

In static analysis for program correctness, the most dominant technique is model checking, which takes a formal model of a finite set of states and automatically proves whether the input specification complies with the model. Solidity contracts have been mostly the target of modeling formal verification is another dominant technique for program correctness. It mathematically proves the correctness of a given formal model against a formal specification. The most common technique in dynamic analysis was fuzzing and runtime verification.

The biggest challenge the paper [5] found was ambiguity in program behaviors arisen from transaction ordering dependency, reliance on off-chain sources, and code opacity. Unstable semantics of programming languages and lack of clear property definitions for vulnerabilities were two other unsolved challenges.

This paper [5] only explores the different research papers related to smart contract analysis. This paper helps us explore the different possibilities in smart contract analysis. This paper [5] aids future researchers about the different available papers and what to look for when analyzing the smart contracts.

This research paper [5] will help us collect the different vulnerabilities which are present in smart contracts such as transaction-ordering dependence, timestamp dependence, mis-handled exceptions, and re-entrancy problems. This will help us set a benchmark for the vulnerabilities to look for while analyzing smart contracts.

This paper “Static Analysis of Integer Overflow of Smart Contracts in Ethereum [6]” by Enmei Lai analyzes the integer overflow of Solidity smart contracts. It summarizes 11 kinds of integer overflow features. After this analysis a static detection tool for integer overflow is designed based on the idea of Smartcheck tool. The designed tool can detect integer overflow vulnerabilities in smart contracts. 7000 smart contracts were tested on the designed tool out of which 430 smart contracts had integer overflow vulnerability.

The paper [6] defines the XPath patterns abstractly based on the vulnerability characteristics above to form a rule library. When the user detects source codes of the smart contract, the tool performs lexical analysis and syntax analysis on the source codes, and generates abstract syntax tree, and then converts it into the

XML-based intermediate representation. Then XPath expression is extracted by traversing the rule library to query and locate the matching nodes in the XML-based intermediate representation. Lastly, the numbers of rows in the source codes are relocated to form the vulnerability analysis report.

The paper [6] evaluates 7000 smart contracts using the designed tool and the accuracy of the detection results is almost 100%. There were 109 smart contracts with multiplication overflow vulnerabilities, 218 smart contracts with addition overflow vulnerabilities and 103 with subtraction overflow vulnerabilities. The paper also compares their tool with other smart contract vulnerability detection tools, but most tools currently fail to detect integer overflow vulnerabilities.

Overall, this paper [6] makes a valuable contribution to the field of Ethereum security by demonstrating the effectiveness of static analysis techniques for detecting and preventing integer overflows in smart contracts. It provides a useful tool for developers to use in securing their contracts, and also serves as a useful reference for researchers interested in this area.

This paper, "SmartCheck: Static Analysis of Ethereum Smart Contracts [7]" by Sergei Tikhomirov provides a comprehensive classification of code issues in solidity and also implements SmartCheck which is an extensible analysis tool that detects issues. SmartCheck is evaluated on a big dataset of real world contracts and then they are compared with manual audits. Symbolic execution involves replacing concrete values in the code with symbolic values and executing the code to explore the different possible paths and outcomes. This allows SmartCheck to test a wide range of input values and uncover hidden vulnerabilities that might not be detected by other static analysis tools.

The paper [7] compiles a list of 4 main categories of issues which include functional issues, security issues, operational issues and developmental issues. There are various issues in each category. A tool is implemented called SmartCheck which is implemented in Java. A number of code issues were detected by SmartCheck such as Balance equality, Unchecked external call and Re-entrancy. A number of similar issues are identified and these are checked with three available tools, Oyente, Remix and Securify on three contracts.

The results show that SmartCheck was able to detect a variety of security vulnerabilities with high accuracy and low false positive rates. As per SmartCheck, 99.9% of contracts have issues, 63.2% of contracts have critical vulnerabilities. The results are displayed in figure 7.

The SmartCheck tool can be improved in multiple directions: improving the grammar, making patterns more precise (e.g., the temporarily muted Unchecked math), adding new patterns, implementing more sophisticated static analysis methods, adding support for other languages. This paper [7] makes a valuable

contribution to the field of Ethereum security by demonstrating the effectiveness of static analysis techniques, particularly symbolic execution, for detecting security vulnerabilities in smart contracts. It provides a useful tool for developers to use in securing their contracts, and also serves as a useful reference for researchers interested in this area.

Severity	Pattern	Findings	% of all
high	Re-entrancy	4015	3.329
	Unchecked external call	986	0.818
	Transfer forwards all gas	275	0.228
medium	DoS by external contract	7864	6.521
	Timestamp dependence	7692	6.378
	`send` instead of `transfer`	3370	2.794
	Costly loop	2610	2.164
	Unsafe type inference	638	0.529
	Using `tx.origin`	197	0.163
	Balance equality	113	0.094
low	Implicit visibility level	81160	67.296
	Compiler version not fixed	3699	3.067
	Integer division	1727	1.432
	Style guide violation	1626	1.348
	`private` modifier	1223	1.014
	Token API violation	1410	1.169
	Malicious libraries	1395	1.157
	Locked money	530	0.439
	Redundant fallback function	64	0.053
	Byte array	7	0.006

Figure 7: Number of code issues detected along with the overall percentage [7]

This paper "Clairvoyance: Cross-contract Static Analysis for Detecting Practical Reentrancy Vulnerabilities in Smart Contracts [8]" by Jiaming Ye presents Clairvoyance, a cross-function and cross-contract static analysis by identifying infeasible paths to detect reentrancy vulnerabilities in smart contract. It presents a large-scale empirical study to evaluate the effectiveness of three recent general-purpose static tools using 11714 real world contracts.

Fig. 8 shows an overview of the approach. In Fig. 8, each rectangle represents an artifact, each ellipse represents a step (or process), each edge represents the workflow of how a step takes some artifacts and outputs some new artifact. As illustrated in the diagram, the system takes Solidity code as input, then outputs reentrancy paths after path collecting and path filtering steps.

To address reentrancy issue, Ye [8] developed Clairvoyance, a static analysis tool that uses interprocedural control-flow analysis to analyze the code of smart contracts and identify potential vulnerabilities. Interprocedural control-flow analysis involves analyzing the control flow of a program across multiple procedures or functions, rather than just within a single procedure. This allows Clairvoyance to analyze the interactions between different contracts and identify reentrancy vulnerabilities that might not be detected by other static analysis tools.

In the findings, the tools are not sufficiently effective in handling the reentrancy bug. The reason is that most of their reports are false negatives. So as to avoid these false positives happening to the tool, the paper [8] audit incorrect reports manually. The experiments are performed on 17770 new contracts obtained from Google Big Query open dataset.

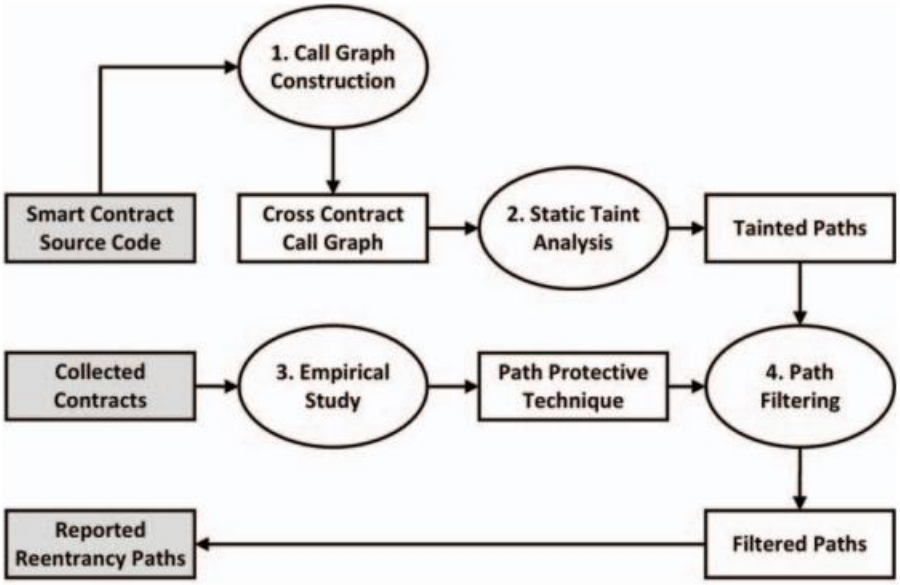

Figure 8: System Diagram. Rectangle represents an artifact, ellipse represents a step, edge represents the workflow [8]

For the numbers of detection results of the four static tools, Slither reports 162 vulnerabilities in total, of which 3 reports are true positives (TPs), while the other 159 reports are false positives (FPs). Oyente has least 28 reports, of which 4 results are TPs, while the rest 24 reports are FPs.

This paper [8] makes a valuable contribution to the field of Ethereum security by demonstrating the effectiveness of static analysis techniques, particularly interprocedural control-flow analysis, for detecting reentrancy vulnerabilities in smart contracts.

The next paper, “SolAnalyser: A Framework for Analysing and Testing Smart Contracts [9]” by Sefa Akca proposes techniques which will allow complete automated analysis of smart contracts, using both static and

dynamic techniques to reduce the number of false positives, and handle the entire syntax of smart contracts. The author injects well known vulnerabilities into smart contracts. Four major works in this paper are static checks, test generator, runtime monitoring, fault seeding tool and empirical evaluation.

The approach for analysis of smart contracts includes three main components, 1) A vulnerability detection technique, SolAnalyser, shown in Fig. 9, that combines static analysis, which is a code instrumentation framework for Solidity, and dynamic analysis. 2) An automated input generation for smart contracts 3) A tool, MuContract, that artifically seeds different types of vulnerabilities in smart contracts. MuContract is used to assess effectiveness of SolAnalyser in revealing the seeded vulnerabilities. They also use the mutated contracts generated by MuContract to assess other existing analysis tools in the literature and compare with SolAnalyser.

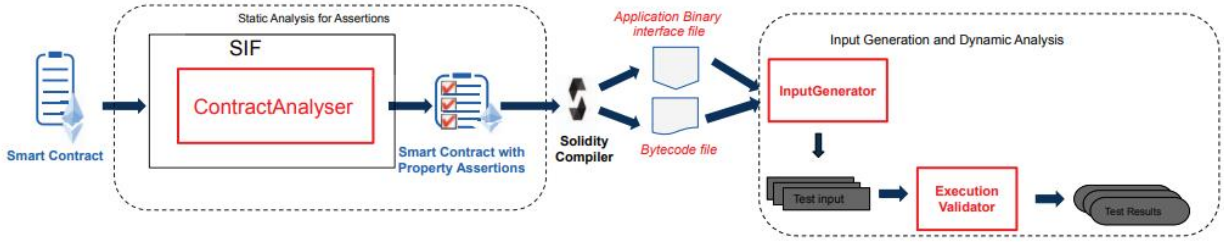

Figure 9: SolAnalyser: Assertion injection, input generation and analysis. Flowchart of the method used [9]

SolAnalyser with static and dynamic checks supported by test generation, was effective at detecting vulnerabilities across all 1838 contracts and the 12866 mutated versions, with a precision of 72% and recall rate of 100%. The technique was capable of detecting more types of vulnerabilities than all five existing analysis tools used in the experiment.

In this paper [9], they have proposed a fully automated technique for vulnerability detection in smart contracts that uses code instrumentation and execution trace analysis. They present detection of 8 vulnerability types that have limited analysis support in literature. The framework for inserting property checks in Solidity code is generic and provides interfaces that can easily be used to support detection of other types of vulnerabilities. They also perform dynamic analysis on the Ethereum virtual machine. There is also support for aritificially seeding different types of vulnerabilities in Solidity contracts. The mutated contracts can be used to assess effectiveness of analysis tools in revealing the seeded vulnerabilities.

The next paper, “RA: Hunting for Re-Entrancy attacks in ethereum smart contracts via static analysis [10]” by Yuchiro Chinen presents a static analysis tool named RA (Re-entrancy analyzer) to analyze smart contract vulnerabilities against re-entrancy attacks. RA supports analysis of inter-contract behaviors by using only the Ethereum Virtual Machine bytecodes of target smart contracts, i.e., even without prior

knowledge of attack patterns and without spending Ether. This paper aims to design an inter-contract static analysis tool that uses only EVM bytecodes as input, eliminates false negatives and false positives, and does not require analysts to have a priori knowledge of the attacks on contracts.

We can use RA to evaluate and analyze smart contract vulnerabilities especially reentrancy issues in our research.

This paper “SmartBugs: A framework to analyze solidity smart contracts [11]” by Joao F. Ferreira present SmartBugs, an extensible and easy-to-use execution framework that simplifies the execution of analysis tools on smart contracts written in Solidity, the primary language used in Ethereum.

One of the main goals of SmartBugs is to facilitate the reproducibility of research in automated reasoning and testing of smart contracts. To demonstrate that integration of new tools and comparison with existing tools is easy, this paper extended SmartCheck and used SmartBugs to show that the extended version improves substantially the detection of vulnerabilities related to Bad Randomness, Time Manipulation, and Access Control.

SmartBugs will help us in the research by detection in different vulnerabilities and false positives mentiones above.

The next paper “A Semantic Analysis-Based Method for Smart Contract Vulnerability [12]” by Xingrun Yan proposes a semantic analysis-based method for the smart contract code in order to effectively defend vulnerability attacks and improve detection efficiency. They implement approach as a self-contained tool, which is then evaluated with 99 smart contracts and compares with other detection tools. The experiment results show that the method can improves the accuracy of vulnerability detection and the ability of vulnerability identification obviously.

Fig 10 provides the overall summary of workflow in this research paper.

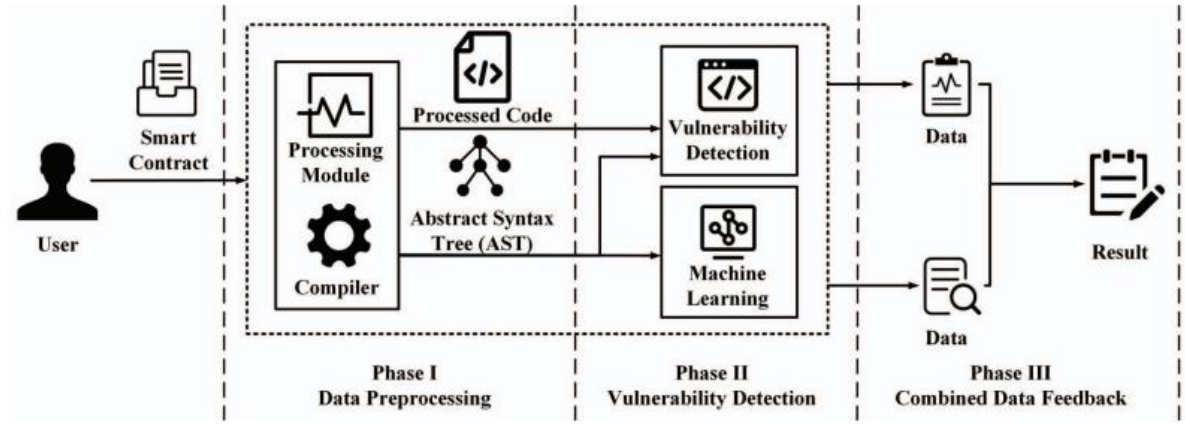

Figure 10: FlowChart of the method used in this paper [12]

To evaluate the effectiveness of the method, they examined four smart contract datasets containing different numbers of different vulnerability types. We compared the method with Mythri and Slither. It could be seen that this method could all accurately detect the number of fallback function reentrancy vulnerabilities and cross-function reentrancy vulnerabilities. However, neither Mythril nor Slither could detect all the data of the two vulnerabilities. Therefore, this method was more effectiveness than Mythril and Slither.

Research Papers	**Summary**
How Effective are Smart Contract Analysis Tools? Evaluating Smart Contract Static Analysis Tools Using Bug Injection By Asem Ghaleb and Karthik Pattabiraman [1].	SolidiFI was used to evaluate 6 smart contract static analysis tools, and the evaluation results show several cases of bugs that were not detected by the evaluated tools even though those undetected bugs are within the detection scope of tools. SolidiFI thus identifies important gaps in current static analysis tools for smart contracts, and provides a reproducible set of tests for developers of future static analysis tools. It also allows smart contract developers to understand the limitations of existing static analysis tools with respect to detecting security bugs.
Empirical Vulnerability Analysis of Automated Smart Contracts Security Testing on Blockchains	The paper tested analysis tools on ten real-world smart contracts from both vulnerability,

By Reza M. Parizi, Ali Dehghantanha, Kim-Kwang Raymond Choo, and Amritraj Singh [2].	effectiveness and accuracy of true detection viewpoints. The results of the experiments showed that SmartCheck tool is statistically more effective than the other automated security testing tools with lowest number of false alarms. The work indicates that research on the empirical knowledge evaluation of security testing for smart contracts is scarce in the literature, especially in relation to internet of things. This work contributed towards filling this gap by providing: (1) a fine-grained methodology to conduct such empirical study for future use by fellow researchers, (2) comparable experimental results on the state-of-the-art smart contracts security testing tools, and (3) statistical tests and constructive insights into the challenges associated with testing smart contracts.
Security Vulnerabilities in Ethereum Smart Contracts By A.Dika and M. Nowostawski [3].	The first taxonomy outlines already exploited vulnerabilities and classifies them based on their architectural and severity level. It serves as a list of issues that can aid developers who plan to develop smart contract applications. The second one is a novel taxonomy of current security tools. The paper has classified the tools based on the methodology they use, the user interface, and the analysis they are able to execute, which allows them to build a 'state of the art' security tools on Ethereum. Lastly, the author construct a matrix of security tools and the vulnerabilities they cover in order to identify gaps and absent vulnerability checks.
Security Vulnerabilities in Ethereum Smart Contracts By Alexander Mense and Markus Flatscher [4].	Looking at the code analysis tools, results of this research work showed that now security tools just cover a certain number of vulnerabilities. Some, such as Remix and Security focus on severe

	vulnerabilities only, while others, such as SmartCheck, Mythril, and Oyente, feature a more comprehensive set of vulnerability detection capabilities. Meanwhile Gasper and F* focus on only one or two very specific vulnerabilities. The majority of the tools are still in beta and several shortcomings were identified. Some of the missing features are highlighting the line containing vulnerable code, explaining detected vulnerabilities, proposing a reference solution and showing simple examples, providing additional resources for each vulnerability, providing an option to mark false positives, showing the total sum of detected vulnerabilities.
Analysis of Blockchain Smart Contracts: Techniques and Insights By S. Kim and S. Ryu [5].	The paper concluded that the biggest challenge is ambiguity in program behaviors arisen from transaction ordering dependency, reliance on off-chain sources, and code opacity. Unstable semantics of programming languages and lack of clear property definitions for vulnerabilities are two other unsolved challenges. They proposed that designing new languages or type systems is a promising research direction, as linguistic supports can ease smart contract analysis. Also, adopting machine learning techniques in this domain is at an early stage with large potentials
Static Analysis of Integer Overflow of Smart Contracts in Ethereum [6]	The paper summarizes 11 different types of integer overflow features. On this basis, an extensible static detection tool was designed to implement the detection of integer overflow of Ethereum smart contract. They tested the tool on 7,000 verified smart contracts and detected 430 smart contracts with vulnerabilities of integer overflow. The tool has the characteristics of high detection efficiency

	and strong expansibility. However, some unknown integer overflow vulnerabilities cannot be detected.
SmartCheck: Static Analysis of Ethereum Smart Contracts [7]	The paper provided a comprehensive overview and classification of the currently known code issues in Solidity – the major high-level language for Ethereum smart contracts. We implemented SmartCheck – an efficient static analysis tool for Solidity. They also tested out tool on a massive set of real-world contracts and detected code issues in the vast majority of them.
Clairvoyance: Cross-contract Static Analysis for Detecting Practical Reentrancy Vulnerabilities in Smart Contracts [8]	They present Clairvoyance, a cross-function and cross-contract static analysis by identifying infeasible paths to detect reentrancy vulnerabilities in smart contracts. To reduce false positives, they have summarized five major path protective techniques (PPTs) to support fast yet precise path feasibility checking. They have implemented and compared Clairvoyance with three state-of-the-art tools on 17770 real-worlds contracts. The results show that Clairvoyance yields the best detection accuracy among all the tools.
SolAnalyser: A Framework for Analysing and Testing Smart Contracts [9]	Present a fully automated technique, SolAnalyser, for vulnerability detection over Solidity smart contracts that uses both static and dynamic analysis. They use the mutated contracts for assessing the effectiveness of different analysis tools. The experiment uses 1838 real contracts from which we generate 12866 mutated contracts by artificially seeding 8 different vulnerability types. They also find that their technique outperforms all five existing tools in supporting detection of all 8 vulnerability types and in achieving higher precision and recall rate. SolAnalyser was also faster in analysing the

	different vulnerabilities than any of the existing tools in the experiment.
RA: Hunting for Re-Entrancy Attacks in Ethereum Smart Contracts via Static Analysis [10]	This paper aims to design an inter-contract static analysis tool that uses only EVM bytecodes as input, eliminates false negatives and false positives, and does not require analysts to have a priori knowledge of the attacks on contracts. They also conducted experiments on deployed contracts and confirmed the performance of RA by precisely identifying combinations of contracts with and without vulnerabilities. The aforementioned performance could be obtained by the high-level combination of the symbolic execution and the Z3 SMT solver.
SmartBugs: A Framework to Analyze Solidity Smart Contracts [11]	This paper presents SmartBugs, an extensible and easy-to-use execution framework that simplifies the execution of analysis tools on Solidity smart contracts. One of the main goals of SmartBugs is to facilitate the reproducibility of research in automated reasoning and testing of smart contracts. SmartBugs is currently distributed with support for 10 tools and two datasets of Solidity contracts. The first dataset can be used to evaluate the precision of analysis tools, as it contains 143 annotated vulnerable contracts with 208 tagged vulnerabilities. The second dataset contains 47,518 unique contracts collected through Etherscan.
A Semantic Analysis-Based Method for Smart Contract Vulnerability [12]	This paper proposed and developed a semantic analysis-based method for vulnerability detection. This approach combined the advantages of semantic analysis-based vulnerability detection with the ability of automate detection. They demonstrated the detection capability of this method by applying it to 99 smart contracts

	simultaneously with two other vulnerability detection tools. The experimental evaluations showed the proposed method had a superior performance. Through the results, we found that these method had certain advantages in vulnerability detection efficiency, and at the same time had a better vulnerability detection accuracy.

Table 4: Summary of Literature Analysis

5 Proposed Approach

This paper proposes to develop a standard benchmark bases on state-of-the-art popular tools, to address false positives, we are to use program analysis techniques – via manual line by line lookup and using symbolic execution.

We will select a set of open source smart contracts, then we will execute these smart contracts on selected analysis tools. We will use the flagged vulnerability reports of these tools to calculate effectiveness and accuracy of our tools. Lastly, we will benchmark the tools on the basis of vulnerabilities they catch.

On the basis of our benchmark, we will be trying to reduce the number of false positives reported by some state-of-the-art popular tool. Next, we incorporate a Machine Learning model to classify a new or unknown tool as per type of problems it may have.

6 Origin Dollar

6.1 Executive Summary

From December 19 2022 through January 9, 2023, we engaged Origin Protocol to review the security of Origin Dollar. During the analysis, several issues were found. The high-severity issues they found were:

- Transaction order dependence
- Random Number
- Unpredictable state
- Callstack depth
- Lost ether
- Reentrancy

- Unchecked send
- Tx.origin
- Blockhash
- Send
- Selfdestruct
- Visibility
- Unchecked send
- Costly pattern
- Bad coding pattern
- Deprecated

Overall, it was found that the Origin Dollar contracts are not yet ready for deployment. The high severity issue that allowed contract funds to be drained, caused by missing input validation and not taking reentrancy into account, exemplifies the current state of the project. Missing input validation in dozens of functions and issues in Governance contracts further indicated that more work is required before deployment. Finally, several issues were detected using automated analysis with Manticore and Mythril, including a high severity vulnerability, highlighting the processes for testing and verification that need improvement.

6.2 Project Dashboard

The following dashboards presents the vulnerability matrix for different tools and false positive as well as false negatives.

Name	Origin Dollar
Type	Solidity
Platforms	Ethereum

Table 5: Application Summary

Total High-Severity Issues	8
Total Medium-Severity Issues	1
Total Low-Severity Issues	6
Total Informational-Severity Issues	5
Total Undetermined-Severity Issues	3
Total	23

Table 6: Vulnerability Summary

Data Validation	9
Undefined Behavior	8
Access Controls	1
Arithmetic	1
Standards	1
Timing	1
Auditing and Logging	1
Denial of Service	1
Total	23

Table 7: Category breakdown

Vulnerability Type	**Mythril**
Dependence on predictable environment variable	3
Multiple calls in a single transaction	2
Assertion violation	2
Arithmetic	1

Table 8: False positives detected

Vulnerability Type	**Manticore**
Bad randomness	1
Arithmetic overflow	2
Denial of service	2
reentrancy	1
Unchecked calls	1

Table 9: True positives detected

7 References

[1] A. Ghaleb and K. Pattabiraman, "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*, Jul. 2020, pp. 415–427. doi: 10.1145/3395363.3397385.

[2] R. M. Parizi, A. Dehghantanha, K.-K. R. Choo, and A. Singh, "Empirical Vulnerability Analysis of Automated Smart Contracts Security Testing on Blockchains." Sep. 07, 2018. doi: 10.5555/3291291.3291303.

[3] A. Dika and M. Nowostawski, "Security Vulnerabilities in Ethereum Smart Contracts," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Halifax, NS, Canada, Jul. 2018, pp. 955–962. doi: 10.1109/Cybermatics_2018.2018.00182.

[4] A. Mense and M. Flatscher, "Security Vulnerabilities in Ethereum Smart Contracts," in *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services*, New York, NY, USA, Nov. 2018, pp. 375–380. doi: 10.1145/3282373.3282419.

[5] S. Kim and S. Ryu, "Analysis of Blockchain Smart Contracts: Techniques and Insights," in *2020 IEEE Secure Development (SecDev)*, Sep. 2020, pp. 65–73. doi: 10.1109/SecDev45635.2020.00026.

[6] E. Lai and W. Luo, "Static Analysis of Integer Overflow of Smart Contracts in Ethereum," in *Proceedings of the 2020 4th International Conference on Cryptography, Security and Privacy*, New York, NY, USA, Feb. 2020, pp. 110–115. doi: 10.1145/3377644.3377650.

[7] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: static analysis of ethereum smart contracts," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, New York, NY, USA, May 2018, pp. 9–16. doi: 10.1145/3194113.3194115.

[8] J. Ye, M. Ma, Y. Lin, Y. Sui, and Y. Xue, "Clairvoyance: Cross-contract Static Analysis for Detecting Practical Reentrancy Vulnerabilities in Smart Contracts," in *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, Oct. 2020, pp. 274–275.

[9] S. Akca, A. Rajan, and C. Peng, "SolAnalyser: A Framework for Analysing and Testing Smart Contracts," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2019, pp. 482–489. doi: 10.1109/APSEC48747.2019.00071.

[10] Y. Chinen, N. Yanai, J. P. Cruz, and S. Okamura, "RA: Hunting for Re-Entrancy Attacks in Ethereum Smart Contracts via Static Analysis," in *2020 IEEE International Conference on Blockchain (Blockchain)*, Nov. 2020, pp. 327–336. doi: 10.1109/Blockchain50366.2020.00048.

[11] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "SmartBugs: A Framework to Analyze Solidity Smart Contracts," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Sep. 2020, pp. 1349–1352.

[12] X. Yan, S. Wang, and K. Gai, "A Semantic Analysis-Based Method for Smart Contract Vulnerability," in *2022 IEEE 8th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, May 2022, pp. 23–28. doi: 10.1109/BigDataSecurityHPSCIDS54978.2022.00015.