CSE 6324 – ITERATION 1

PROJECT: TESTING AUCTION SMART CONTRACTS FOR ETHEREUM VIRTUAL MACHINE

Document: Iteration 1 Deliverables

Prepared By:

Venkata Ashok sairam Dasari – 1002029845,

Vankireddy Karthik Reddy - 1002038114

Teja Sankara - 1002033648

Mithun Ekanandan - 1001843906

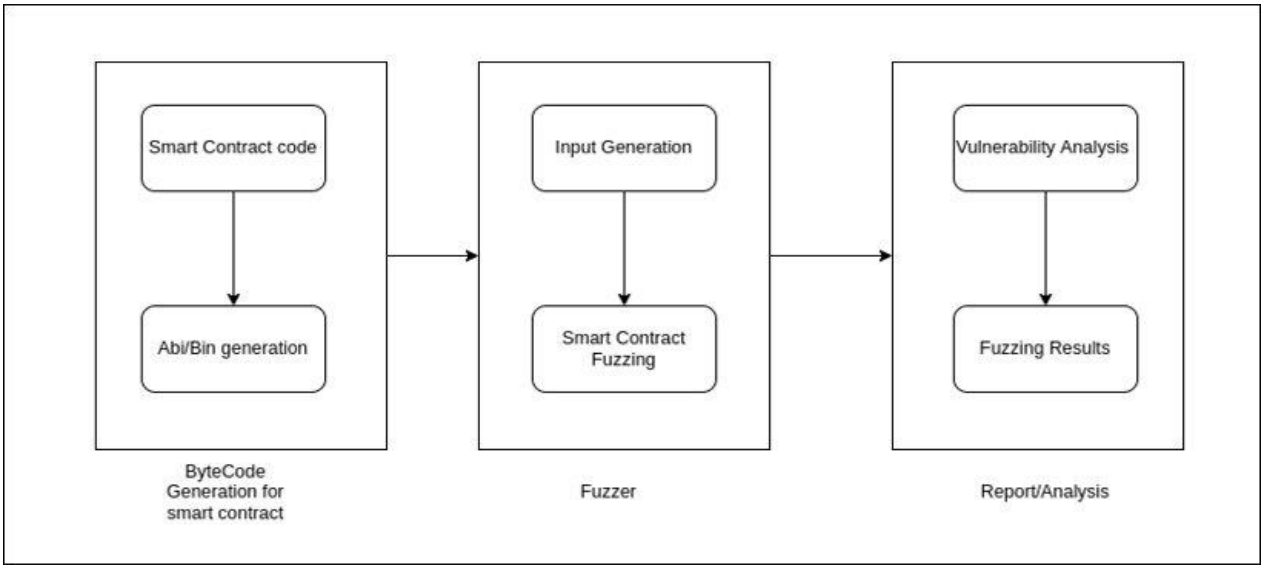
1. Introduction

Auction smart contract allows to place bids and closes the auction at a specific time. Bidders take part in auctions to place bids for a product, the highest bidder is the winner. This smart contract helps to avoid third party auctioneers conducting auctions and saves money by avoiding paying money to them to conduct auctions.

Fuzz testing helps to input unexpected or invalid test inputs to the smart contract to test them to detect any vulnerabilities in the code. Fuzzing checks the negative results to the inputs which identifies the performance and quality issues in the code.

2. Specification and Design

Architecture:



Smart contract code: Auction Smart contract code is input for this fuzz testing tool.

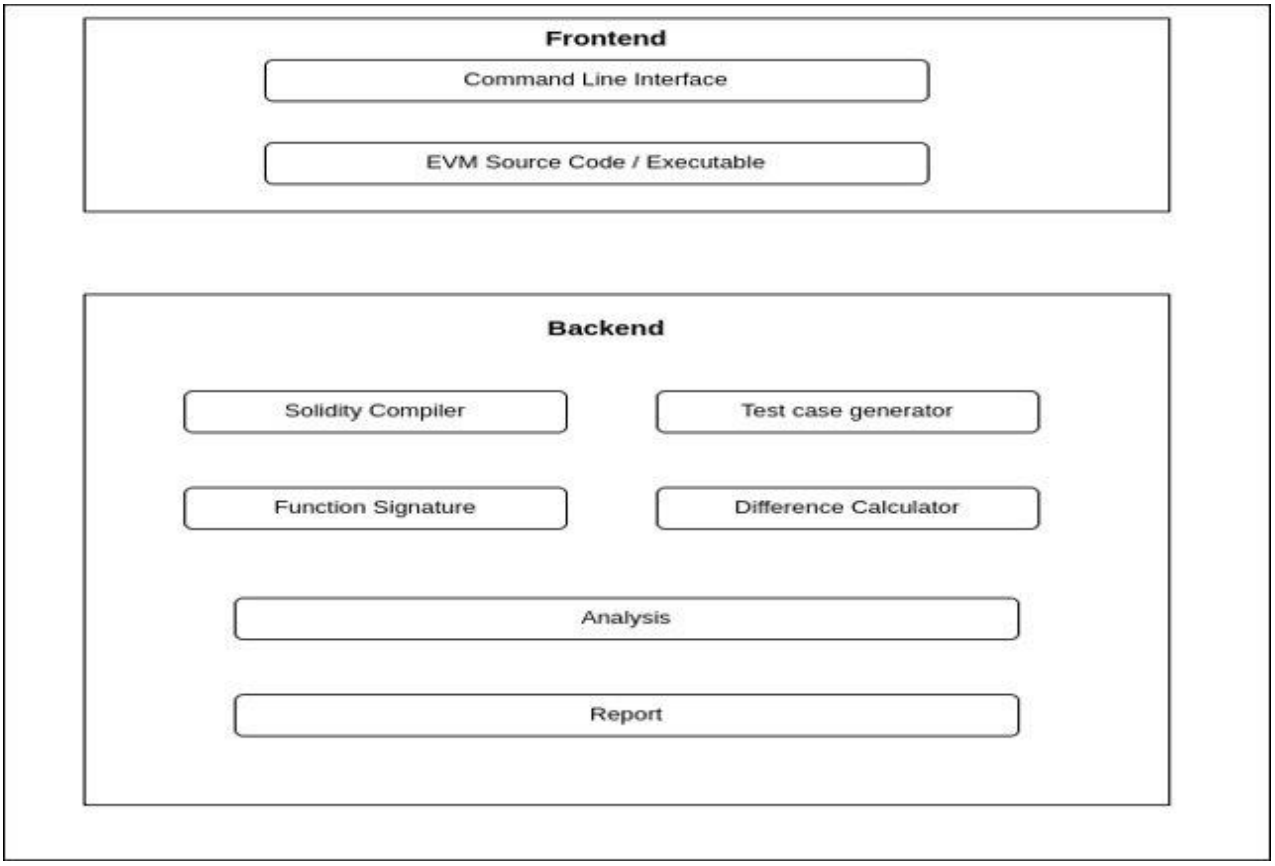
Abi generation: The Auction smart contract will be compiled into Abi format which is in the form of JSON. The ABI has the information about the caller of the function which encodes the information like function signatures and variable declarations.

Input Generation for Abi: Based on the ABI, fuzzing inputs will be generated. For E.g., if variable is an integer, the generated inputs will be maximum of 128 integer type.

Smart contract Fuzzing: Fuzzer tools help to find the vulnerabilities in the smart contract by covering all the pre-defined scenarios.

Vulnerability Analysis: Now the vulnerabilities from smart contract fuzzing are to be analyzed and to be reported promptly.

Report: The report contains the number of failed test cases, number of lines of code in the smart contract and at which line the test case failed with what error.



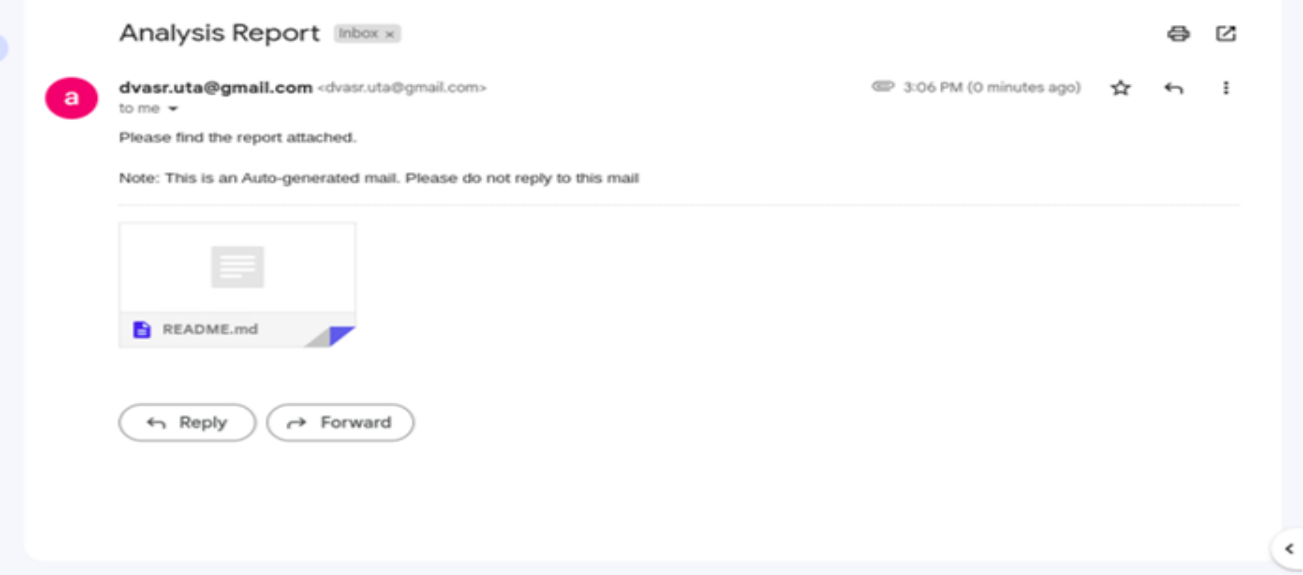
3. Project Plan

Module	Iteration	Time Estimate (In days)	Status
Smart Contract Implementation	**Iteration 1**	**4**	**Completed**
Mail notification feature implementation	**Iteration 1**	**3**	**Completed**
Boiler plate Setup	Iteration 2	6	In-Progress
Test case / Seed Generation Module	Iteration 2	10	Backlog
Fuzzer Module	Iteration 3	10	Backlog

Analysis report module	Iteration 3	9	Backlog
Bug Fix in Project Module	Before Final Deliverable	14	Backlog
Result Comparison between tools	Before Final Deliverable	5	Backlog

4. Iteration 1 Feature Implementation

Feature 1: Implemented mail notification code to send reports to the user. This code will send report to the user after the execution of the results in a tabular/json format [3].



Feature 2: Implemented the logic for Simple Auction, Dutch Auction, Addition and Multiplication smart contract which will be tested upon tool built in the next phases [3].

5. Risks

Risk 1:

Third party Libraries: Using un-tested third-party libraries in the project module increases the vulnerability risks[1].

Mitigation: Always use updated third party libraries which are tested and from a valid resource (For Eg: npmjs, pypi, etc..).

Risk 2:

Faulty EVMs: EVMFuzzer, a tool, which measures the internal difference in execution information between EVMs. They found 5 unreported vulnerabilities in various EVMs. Even Faulty EVMs also poses risks [2].

Mitigation: Decide the EVM with lowest vulnerabilities for building a fuzzing tool because EVMs are also posing risks.

Risk 3:

False Negatives:

Risk 4: Fuzzing has higher false negatives when compared to symbolic execution tools because of the lower coverage for fuzzing when compared to symbolic execution.

Mitigation: Tools like oyente, Echidna have lower false negatives when they tested the tools with lot of smart contracts because of wide range of code coverage. Similar implementations will help to reduce this risk.

Wide range code coverage: A Fuzzing tool said to be complete only when it has wide range of code coverage to find the vulnerabilities in the smart contract code because smart contract is not like a normal program, once deployed in the production it is less likely to change the code.

Mitigation: Test the smart contract solidity/vyper code with various fuzzing tools to find the logical bugs in the code.

6. Customers and Users

Smart Contract Developers: This tool provides CLI for developers to test smart contracts.

Organizations: As the tool is yet to be built and tested, we hope in the future this tool will be used by organizations who are more specific in building auction smart contracts.

7. References

[1] https://techbeacon.com/security/third-party-libraries-are-one-most-insecure-parts-application

[2] Ying Fu, Meng Ren, Fuchen Ma, Heyuan Shi, Xin Yang, Yu Jiang, Huizhong Li, Xiang Shi. 2019. EVMFuzzer: Detect EVM Vulnerabilities via Fuzz Testing

[3] https://github.com/ashokdv/CSE-6324-Smart-Contracts-for-EVM