Inception

Team 8

Akshata Raikar

Devyani Singh

Ravi Rajpurohit

Shovon Pereira

[CSE 6324-001]

Introduction

Blockchain technology is a ground-breaking technology that allows users to communicate without needing a trusted intermediary [1]. A "smart contract" is simply a program that runs on the blockchain. It is a collection of code (its functions) and data (its state) that resides at a specific address [2]. Smart contracts automate the transaction of valuable financial assets. Any kind of error in smart contracts may result in huge financial losses. That is why it is important to test smart contracts thoroughly before deploying them to the blockchain. Smart contracts deployed in Ethereum Blockchain are immutable; therefore, it is imperative to look at vulnerabilities before deploying.

Since each Ethereum transaction requires computational resources to execute, they must be paid for to ensure Ethereum is not vulnerable to spam and cannot get stuck in infinite computational loops. Payment for computation is made in the form of a gas fee [3]. To ensure efficient gas usage it is important to know about excessive use of gas, inefficient code patterns, and precise gas usage amount of solidity functions. Our proposed tool intends to address all these concerns.

Repository

https://github.com/PereiraMavs/CSE6324_Team_8

Version: 0.1

Vision

Smart Contracts with inefficient code can waste resources. Also executing a code that surpasses the gas limit can cause exceptions and halt the program midway. It is always good to look for these gas related vulnerabilities beforehand. Many static and dynamic analysis tools have been developed to detect this kind of situation [4][5].

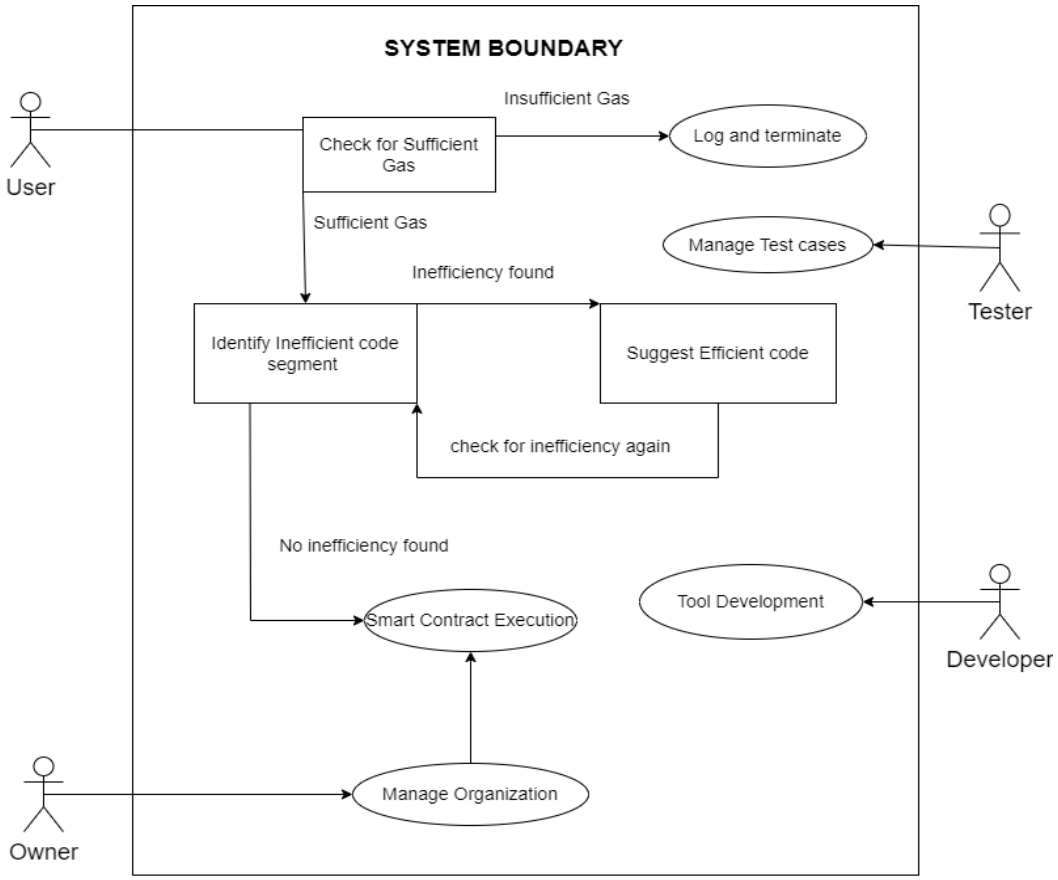
There are other vulnerabilities that threaten the safety and security of smart contracts. Among many other tools that detect security risks in smart contracts, Slither is one such tool that uses static analysis techniques to detect almost 88 types of threats. It also supplies different APIs to build other analysis tools using its intermediate representation [6].

The vision of this project is to use Slither to build a unified tool that will detect excessive gas usage and find inefficient codes that waste gas. GasGauge [7] and GASOL [8] tools do something similar but if merged with Slither it will add significant value and make a more powerful tool.

Proposed Features

- Estimate gas cost of a smart contract function before execution to ensure it does not exceed the limit.
- Analyze code to detect inefficient code so that it can be changed to save gas.
- The codes of the tool will be found in the attached GitHub repository.
- The tool will be console based.

Flow Diagram



User Stories

Entity	Objective	Desired Outcome
User	Do secure transactions	Efficient gas usage.
Developer	Develop a tool that can effectively do static and dynamic analysis and can do the transactions efficiently ensuring integrity.	Develop efficient and desired end product.
Owner	Sell an efficient and correct tool.	To not lose customers.
Tester	To search for vulnerabilities in smart contract programs.	To ensure that the program being deployed has no vulnerabilities.

Should/Could/Won't have features

If time allows the tool will have a feature to detect gas value of individual line of code.

The tool will be able to find out inefficient code, suggesting alternative coding strategies could be added.

The tool will not have any user interface, it will be totally used from the console.

Customers

- **Dr. Christoph Csallner:** will provide advice about any issues and concerns. Also, he will be providing feedback on the project.
- **Mohammed Rifat Arefin:** will also aid with any issues and feedback. After every iteration, his feedback will help improve deliverables and optimize codes.
- **Team 9:** will review the deliverables and the tool and comment on how the tool can be improved.

Users

- **CSE 6324-001 class:** They are the target audience of this project. They will use the tool and share their feedback on which project can be improved.

Software Development Plan

The goal of the project is to enhance slither for static code analysis by warning users of code inefficiency regarding gas usage. As we know gas is important and should be optimized for better programs, it will be a valuable addition to slither.

The team aims to bring together functionalities from GasGauge and similar tools to expand the capabilities of slither and overall improvement as a tool.

The timeline to complete the development and execution testing is around two months and the project is planned in total three iterations.

- **Iteration 1:** This iteration involves defining a plan for the project's development. We need to clearly define the features boundaries for this project. The reference tools – Slither, GasGauge and others if needed, will be studied by each team member. The team's schedule needs to be planned for this project and each member will take on specific responsibilities. By the end of this iteration, we will start progressing towards a prototype.
- **Iteration 2:** In this iteration, we plan to make a working prototype exploring the defined functionalities. With a prototype the team can collaboratively refine the project requirements and scope.
- **Iteration 3:** By the third iteration, we plan to complete the development phase. The team means to complete all testing and analysis to observe the final results of this project by the end of this phase.

Risk Management Plan

- **Technical Risk:** As we plan to try combining the functions of three different tools into a single program, it is a risk that the process might throw some unanticipated errors that will need some time to be corrected.
- **Resource Risk:** Most of the team lacks familiarity and is inexperienced with Smart Contracts to work smoothly on such a project. It might take some extra time than a standard plan to fill this gap.
- **Schedule Risk:** All the team members have different class schedules and varied commitments. Finding a disciplined yet flexible schedule is a challenge due to conflicting classes and assignments.

Risk Mitigation Plan

- **Technical Risk Mitigation:** We plan to take help from the wide and supportive python community to tackle technical risks. By following the best coding practices, we will reduce the odds of errors and increase the chances of success as defined for the project description.
- **Resources Risk Mitigation:** As we already know about lack of familiarity with such projects, we will expand our knowledge to get better at processing ideas for smart contract based programs. We plan to give extra time to catch up with the inexperience.
- **Schedule Risk Mitigation:** As a team, we plan to be flexible with the project development schedule for each member. The plan includes keeping coherent and regular communication throughout the semester. We believe that keeping a slack of time for each stage will be beneficial in dealing with unexpected issues.

Risk Exposure

Considering each risk category, we believe that Technical and Resource based risks will take the most time to resolve. While Schedule based risks are significant too, its weightage is expected to be lesser than other categories.

- Technical Risk Exposure
  - Occurrence probability (PR) - 0.5
  - Occurrence effect (ER) - Extra 15 hours
  - Exposure of risk – 0.5 (PR) * 10 (ER) = **7.5 hours**
- Resource Risk Exposure
  - Occurrence probability (PR) - 0.5
  - Occurrence effect (ER) - Extra 12 hours
  - Exposure of risk – 0.5 (PR) * 10 (ER) = **6 hours**
- Schedule Risk Exposure
  - Occurrence probability (PR) - 0.2
  - Occurrence effect (ER) - Extra 8 hours
  - Exposure of risk – 0.5 (PR) * 10 (ER) = **1.6 hours**

Constraints

Project Delivery Schedule -

- The written reports, presentations, and repository evaluations need to be prepared for submission at the conclusion of each iteration.
- The submissions should be primed for assessment by Dr. Christopher Csallner, GTA Mohammed Rifat, and Review Team 9. They will evaluate the quality and provide feedback for enhancements in the later iteration.
- Any delays in submitting these materials could lead to postponements in the review process, grading, and feedback, ultimately risking the timely completion of the project as per the final scheduled delivery date.

Competitors

Tool	Features
GasGauge	GasGauge focuses on gas related vulnerabilities that is occurred by out-of-gas situation
GasFuzzer	It improves on ContractFuzzer tool to find out vulnerabilities via gas allowance manipulation
GasChecker	Suggests efficient smart contract coding technique to improve gas usage

References

[1] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur and H. -N. Lee, "Ethereum Smart Contract Analysis Tools: A Systematic Review," in IEEE Access, vol. 10, pp. 57037-57062, 2022, doi: 10.1109/ACCESS.2022.3169902.

[2] https://ethereum.org/en/developers/docs/smart-contracts/ , accessed on 09/23/2023

[3] https://ethereum.org/en/developers/docs/gas/ , accessed on 09/23/2023

[4] T. Chen et al., "GasChecker: Scalable Analysis for Discovering Gas-Inefficient Smart Contracts," in IEEE Transactions on Emerging Topics in Computing, vol. 9, no. 3, pp. 1433-1448, 1 July-Sept. 2021, doi: 10.1109/TETC.2020.2979019.

[5] I. Ashraf, X. Ma, B. Jiang and W. K. Chan, "GasFuzzer: Fuzzing Ethereum Smart Contract Binaries to Expose Gas-Oriented Exception Security Vulnerabilities," in IEEE Access, vol. 8, pp. 99552-99564, 2020, doi: 10.1109/ACCESS.2020.2995183.

[6] https://github.com/crytic/slither, accessed on 09/23/2023

[7] https://arxiv.org/abs/2112.14771, accessed on 09/23/2023

[8] Albert, E., Correas, J., Gordillo, P., Román-Díez, G., Rubio, A. (2020). GASOL: Gas Analysis and Optimization for Ethereum Smart Contracts. In: Biere, A., Parker, D. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2020. Lecture Notes in Computer Science (), vol 12079. Springer, Cham. https://doi.org/10.1007/978-3-030-45237-7_7