Iteration 1

Team 5
Team Members: Abbinav H. Burhanuddin C.
Dhairya C. Ritesh M.
CSE 6324-001

Project Plan

Features Plan

- Understand Gas Gauge and Manticore Tools' Functionality
  - The first goal for the team's feature plan was to first understand the individual features of Gas Gauge and Manticore. By looking at the code structure, the team tried to understand the features of the tools, but due to limited to no comments, the code was not easy to decipher. The team was able to understand a few parts of the code that was specific to Linux OS and decided to convert the code to be compatible with Windows in the Gas Gauge Tool. The team is continuing to process the Manticore Tool and was able to get the Manticore compiled and running with a sample smart contract.

- Have Gas Gauge and Manticore Tools' Compile and Running
  - The second goal for the team's feature plan was to have Gas Gauge and Manticore compiled and running. Both Gas Gauge and Manticore had issues compiling as some libraries for Gas Gauge were outdated. Manticore libraries were more up-to-date than Gas Gauge but still had more compile errors. Gas Gauge was able to compile and run after a few code modifications on Windows OS. Manticore is currently compiling and running on Linux.

- Use Available Smart Contracts as Input for Gas Gauge and Manticore Tools'
  - The third goal for the team's feature plan was to have Smart Contracts available to run through Gas Gauge and Manticore. Gas Gauge Github provided a single contract for its tool, but finding other smart contracts for Gas Gauge and Manticore has proven to be challenging for the team. MantiCore has provided us with very simple and basic contracts for testing, but nothing too complex. Therefore, the team has requested assistance from Shovon in search of smart contracts and he has provided generic sample smart contracts for the team to use while he tracks down Gas Gauge and Manticore-specific smart contracts.

App Differences

- MPro: Incorporates Static Analysis and Symbolic Execution
  - MPro uses both static analysis and symbolic execution to check for bugs within Ethereum Smart Contracts. MPro builds on the symbolic execution of Mythril-Classic and adds a heuristic symbolic execution to the tool. Slither is used to statically analyze the smart contract and then is run through the Mythril-Classic plus heuristic symbolic

    execution feature of MPro to help maximize efficiency of finding bugs in smart contracts while decreasing the vulnerability of smart contract exposures. [1]

- GasCore+: Incorporate Static Analysis, Dynamic Analysis and Symbolic Execution from Gas Gauge and Manticore.
    - GasCore+ uses both static and dynamic analysis and then runs symbolic execution to help enhance the capabilities of finding bugs in Ethereum Smart Contracts. Unlike MPro, GasCore+ uses the static and dynamic analysis efficiency of Gas Gauge to help analyze smart contracts and then is run through MantiCore's symbolic execution process to help eliminate as many vulnerabilities in the Ethereum Smart Contract as possible. The extra dynamic analysis of Gas Gauge provides an extra level of analysis that MPro does not provide, making GasCore+ an all-rounder in Fuzz Testing and Symbolic Execution. [1]
    -

5 Biggest Risks

- Risk 1: No Extra Documentation for Gas Gauge and Manticore.
    - Gas Gauge and Manticore have limited and outdated documentation regarding the installation of the tools and how code is structured in each file. Limited to no comments are provided in each Python file.
    - **Risk Exposure:** The likelihood of Risk 1 occurring is 40% and the impact on this iteration will be about 10 hours spent. The Risk Exposure for Risk 1 is about spending 4 extra hours towards resolving it.
    - **Mitigation Plan:** To help reduce the likelihood and impact of Risk 1, the team found a slack channel for Manticore where users can ask questions regarding the installation process. The team also found YouTube Videos on the installation process for Gas Gauge.

- Risk 2: Gas Gauge running on Windows/Linux OS and Manticore running on LinuxOS
    - Gas Gauge has outdated libraries in the online Github which can cause issues while running it with the current version of python. While Manticore's libraries are kept up-to-date, the lack of documentation can cause issues installing the tool on Linux OS.
    - **Risk Exposure:** The likelihood of Risk 2 occuring is 50% and the impact on this iteration will be about 8 hours spent. The Risk Exposure for Risk 2 is about spending 4 extra hours

towards resolving it.
    - **Mitigation Plan:** The team can contact Shovon for assistance to install Manticore and Gas Gauge on our Windows/Linux OS. Manticore also has a slack channel that can assist with questions regarding the installation of the tool or functionality clarification.

- Risk 3: Finding Available/Compatible Smart Contracts
    - Gas Gauge and Manticore have some specific smart contracts that run well with each individual tool respectively, but finding those specific smart contracts can be difficult due to limited usage of the tool by users.
    - **Risk Exposure:** The likelihood of Risk 3 occurring is 40% and the impact on this iteration will be about 10 hours spent. The Risk Exposure for Risk 1 is about spending 4 extra hours towards resolving it.
    - **Mitigation Plan:** The team can ask Shovon for assistance in finding Gas Gauge and Manticore-specific smart contracts. The team can also contact the Manticore slack channel to ask members for locations of Manticore-compatible smart contracts for testing of the tool.

- Risk 4: Availability of Linux OS Device
    - Team members do not have access to a Linux device or a laptop that has virtual machine capabilities.
    - **Risk Exposure:** The likelihood of Risk 4 occurring is 25% and the impact on this iteration will be about 1 hour spent. The Risk Exposure for Risk 4 is about spending 0.15 extra hours towards resolving it.
    - **Mitigation Plan:** The team members who do not have a Linux device could download and use a virtual machine box. Team members who do not have a laptop with virtual machine capabilities can rent a laptop from the UTA Central Library.

- Risk 5: Team Member Scheduling Issues
    - Team members have other class homework assignments, projects, and quizzes to work on during the current iteration.
    - **Risk Exposure:** The likelihood of Risk 5 occurring is 25% and the impact on this iteration will be about 2 hours spent. The Risk Exposure for Risk 5 is about spending 0.5 extra hours towards resolving it.
    - **Mitigation Plan:** The team members can work with the team to reschedule meetings and reallocate iteration tasks to team

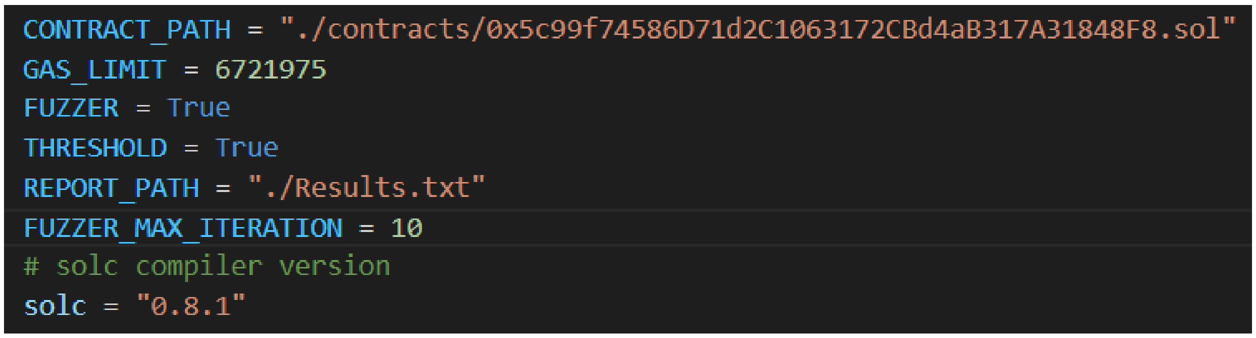
members who can handle a heavier load this iteration than team members who are busy with other class work with similar due dates as the current iteration.

Specification and Design

Input:

The input to the GasCore+ is similar to GasGuage in which we will provide the user the opportunity to define the below inputs:

CONTRACT_PATH - The path where the contract to be analyzed is present.
GAS_LIMIT - The Gas Limit that the user can provide.
FUZZER - An option for users to enable/disable Fuzzer-based testing.
THRESHOLD - An option for users to enable/disable Threshold based testing.
REPORT_PATH - The path where the report of the analysis will be generated.
FUZZER_MAX_ITERATION - The max iterations of the analysis being done.
solc = "0.8.1" - The version of Solidity is based on the user's preference.



```
CONTRACT_PATH = "./contracts/0x5c99f74586D71d2C1063172CBd4aB317A31848F8.sol"
GAS_LIMIT = 6721975
FUZZER = True
THRESHOLD = True
REPORT_PATH = "./Results.txt"
FUZZER_MAX_ITERATION = 10
# solc compiler version
solc = "0.8.1"
```

Output:

Prior to running GasCore+, the user has to specify the below configurations:

output_folder - The folder where the output of the analysis will be stored.
output_file - The name of the txt file that will be created.
gas_gauge_dir - Directory where the contract output will be stored.
new_contract_name - The name of the contract that will be stored after the analysis.

The above configurations help in analyzing the smart contracts and provide us with an output of the fuzzing testing and show a list of issues that a contract might have.

Implemented Feature:

The first use case was to ensure that GasGuage runs on a Windows machine. The current version of GasGuage is built to be run directly on a Linux machine. The logic within GasGuage utilizes Linux commands and libraries to run the tool. The team has modified the GasGuage tool by modifying import statements, upgrading libraries, and changing Linux commands being run through Solidity and Slither.

Future Features:

The first feature from MantiCore we would like to incorporate within GasCore+ would be the “manticore-verify”. Manticore verifier is able to analyze a Solidity contract and detect and test its properties. We would prefer to implement functionalities such as:

- User Contracts - Allows us to define the account of the developer, sender, and psender of the contract which aids in security and prohibits several security vulnerabilities.
- Stop Conditions - Ensures that the exploration of contracts would be stopped after reaching certain exit conditions. We can dictate the maximum number of transactions or maximum coverage which would improve the efficiency of the smart contract analysis.

Code and Tests

The code changes done as part of this iteration are available through our public repository - https://github.com/Abbinav/CSE-6324-Team5/tree/master

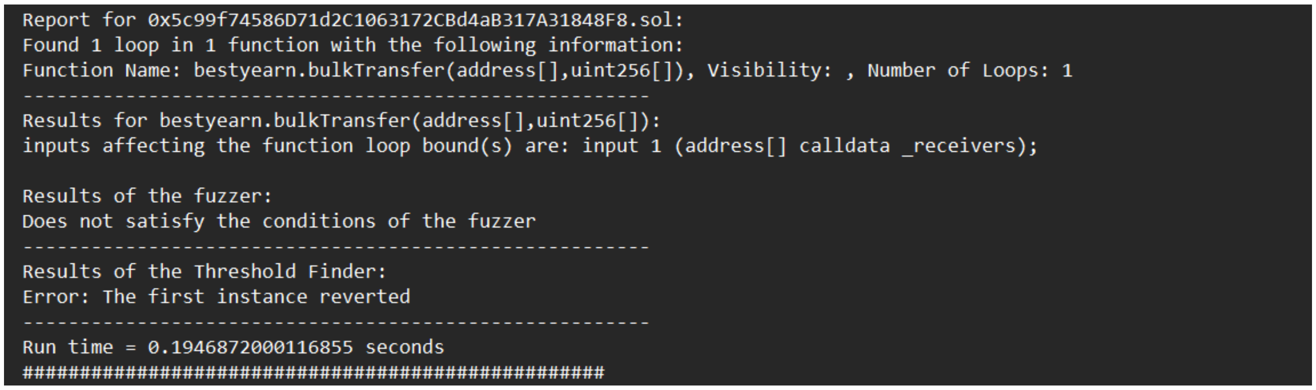
Gas Gauge Code Changes Done:

- Modifying Linux libraries to Windows
- Removing unwanted/outdated imports
- Modifying the Linux commands to analyze the Solidity to Windows commands
- Adding additional code to aid with the Windows implementation

The steps required to run and test the code would be as follows:

- Download the code from the repository above.
- Install the libraries provided within the Readme file of the repository.
- Either utilize the sample contract provided within the project or add your own preferred contract to be analyzed:

  - Modify the contract path as well as the results path within the run.py.
- Run the project using the command “python run.py”.
- After running the project, a text file would be generated within the results path provided with information as shown below.

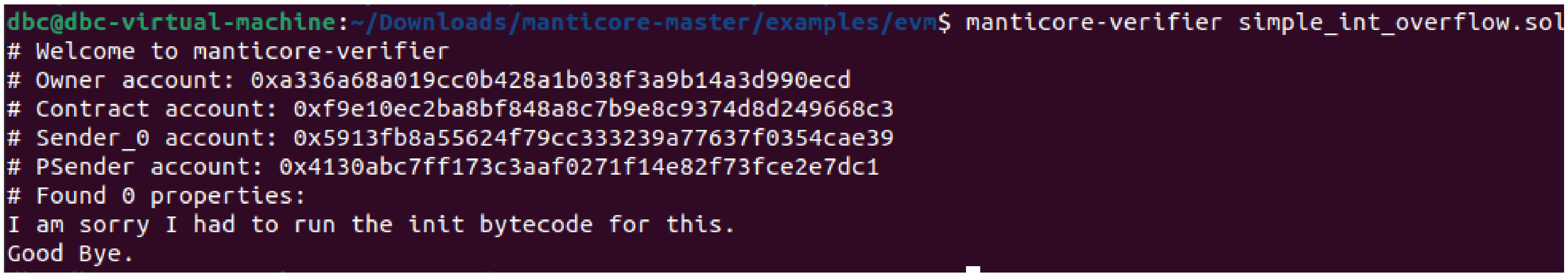

```
Report for 0x5c99f74586D71d2C1063172CBd4aB317A31848F8.sol:
Found 1 loop in 1 function with the following information:
Function Name: bestyearn.bulkTransfer(address[],uint256[]), Visibility: , Number of Loops: 1
-------------------------------------------------------
Results for bestyearn.bulkTransfer(address[],uint256[]):
inputs affecting the function loop bound(s) are: input 1 (address[] calldata _receivers);

Results of the fuzzer:
Does not satisfy the conditions of the fuzzer
-------------------------------------------------------
Results of the Threshold Finder:
Error: The first instance reverted
-------------------------------------------------------
Run time = 0.1946872000116855 seconds
###################################################
```

Manticore Code Changes Done:

- Changes to simple_int_overflow.sol(Smart Contract).
- Changes to solc.py.
- Changes to types.py.

The steps required to run and test the code would be as follows:

- Manticore will run on Linux only.
- Download and install Manticore on Ubuntu on the virtual machine.
- Use the sample smart contract provided with the Manticore or any preferred contract.
- Run Manticore using command /manticore-master/examples/evm$ manticore-verifier (Contract name) on the Linux terminal.
- After running the project, the output will be displayed on the Linux terminal.


```
dbc@dbc-virtual-machine:~/Downloads/manticore-master/examples/evm$ manticore-verifier simple_int_overflow.sol
# Welcome to manticore-verifier
# Owner account: 0xa336a68a019cc0b428a1b038f3a9b14a3d990ecd
# Contract account: 0xf9e10ec2ba8bf848a8c7b9e8c9374d8d249668c3
# Sender_0 account: 0x5913fb8a55624f79cc333239a77637f0354cae39
# PSender account: 0x4130abc7ff173c3aaf0271f14e82f73fce2e7dc1
# Found 0 properties:
I am sorry I had to run the init bytecode for this.
Good Bye.
```

Customers and Users

- Reached out to Shovon for available Smart Contracts
    - The team contacted Shovon for assistance to help locate a few Smart Contracts that can be used to test Gas Gauge and Manticore. Shovon is currently helping the team find smart contracts to use for testing Gas Gauge and Manticore functionality. Shovon provided the team with Generic Smart Contract Github Links for usage. He is currently tracking down Gas Gauge and Manticore specific smart contracts for the team.

- Reached out to Shovon for validation of Gas Gauge results
    - The team contacted Shovon for assistance in validating the Gas Gauge tool's results. Gas Gauge Github provided a smart contract to use. The team ran the smart contract, but the results were unclear to the team. Shovon responded to our request and will look at the results of Gas Gauge and help explain the results validity.

- Reached out to Manticore's Slack Channel for Assistance/Feedback
    - The team contacted members' of the Manticore Slack Channel for assistance in setting up the tool. The Manticore Slack Channel asked for the compile problems and will respond as soon as their team finds a solution.

References

[1] W. Zhang, S. Banescu, L. Pasos, S. Stewart and V. Ganesh, "MPro: Combining Static and Symbolic Analysis for Scalable Testing of Smart Contract," in 2019

[2]https://www.vectorstock.com/royalty-free-vector/agenda-list-concept-vector-13201810, Accessed 9/2022

[3] Risk Management Plan PNG and Risk Management Plan Transparent Clipart Free Download. - CleanPNG / KissPNG, Accessed 09/2022

[4]Computer Icons Customer Service Users' Group PNG, Clipart, Black, Black And White, Business, Circle, Company Free PNG Download (imgbin.com), Accessed 9/2022

[5]https://twproject.com/blog/choose-project-management-software-mother-tongue, Accessed 9/2022

[6]Linux vs. Windows: What's the difference in 2022? | Linux Journal, Accessed 9/2022

[7]https://github.com/gasgauge/gasgauge.github.io, Accessed 9/2022