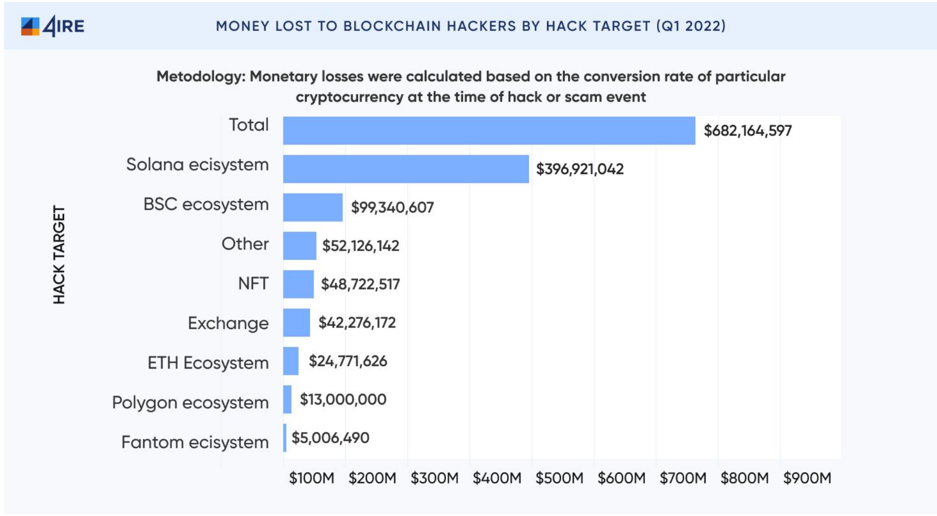
Smart Contract Testing and Analysis for Smart Contract Based Blockchain Application Development

Amir Pirhosseinloo
9 Jan 2024

Overview

- What is smart contract?
  - some use cases: crowdfunding
- Why do we need to test?
  - disasters: millions of losses
- What are the limitation?
  - immutability
  - gas fee

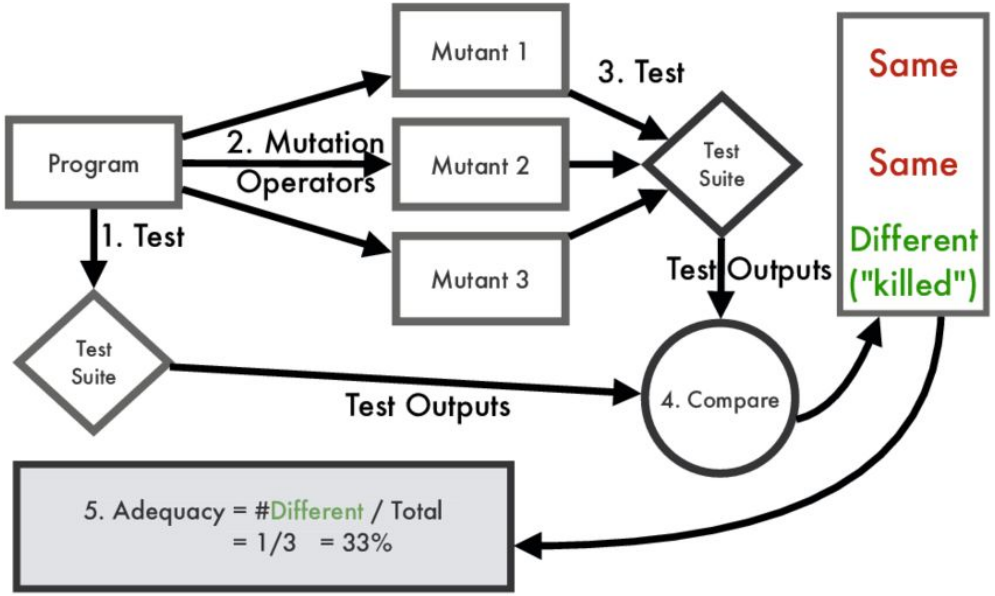


- paper focus: mutation testing, tools
  - code coverage, error seeding, and mutation analysis

Smart Contract Testing (done)

1. Gathered 20 kinds of defects that are malicious and getting rid of them improves the goodness of smart contracts.
2. Generation of effective test cases based on data flow analysis from the constructed control flow graph.
   - genetic algorithm for optimizing the dynamic test of smart contracts
   - obtain the required statements
   - get the definition-use pairs for the Solidity programs.
   - to generate the test case, a genetic algorithm with a function to calculate the definition-use pairs.
3. Contract Fuzzer
   - Of about 6991 Smart Contracts, the tool announced in excess of 459 flaws with high precision, particularly those of "The DAO" bug and the Parity Wallet bug that prompted millions of dollars in misfortunes.
4. ...

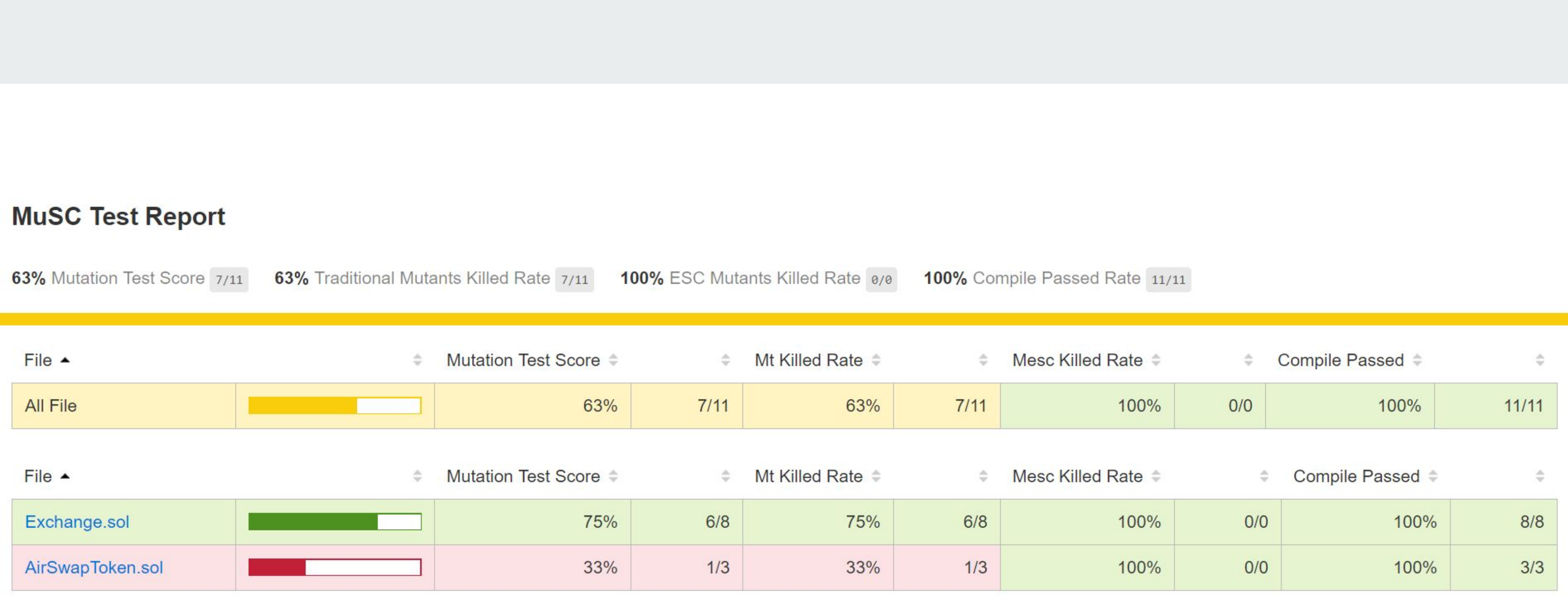
Mutation Testing



Mutation Testing Tools 1: MuSC

https://github.com/belikout/MuSC-Tool-Demo-repo

The author preferred mutation testing to assess the testing's adequacy. The paper discusses in detail the workflow of the mutation testing tool. The smart contract source code written in solidity is first converted into an abstract syntax tree format. The input to the tool is the smart contracts and the test cases. The users select the mutation operators to be used in the tool. Once the operators are selected, the source code is transformed into AST format. The mutants are introduced into the AST formatted source code. During the execution of mutants, the tool records each variant's information as well as the corresponding results of execution. Mutants causing compilation errors are not considered in testing. The mutation score and test results for every mutant are generated as resultant test report. The tool is effective in error injection.



MuSC Test Report

63% Mutation Test Score `7/11` **63%** Traditional Mutants Killed Rate `7/11` **100%** ESC Mutants Killed Rate `0/0` **100%** Compile Passed Rate `11/11`

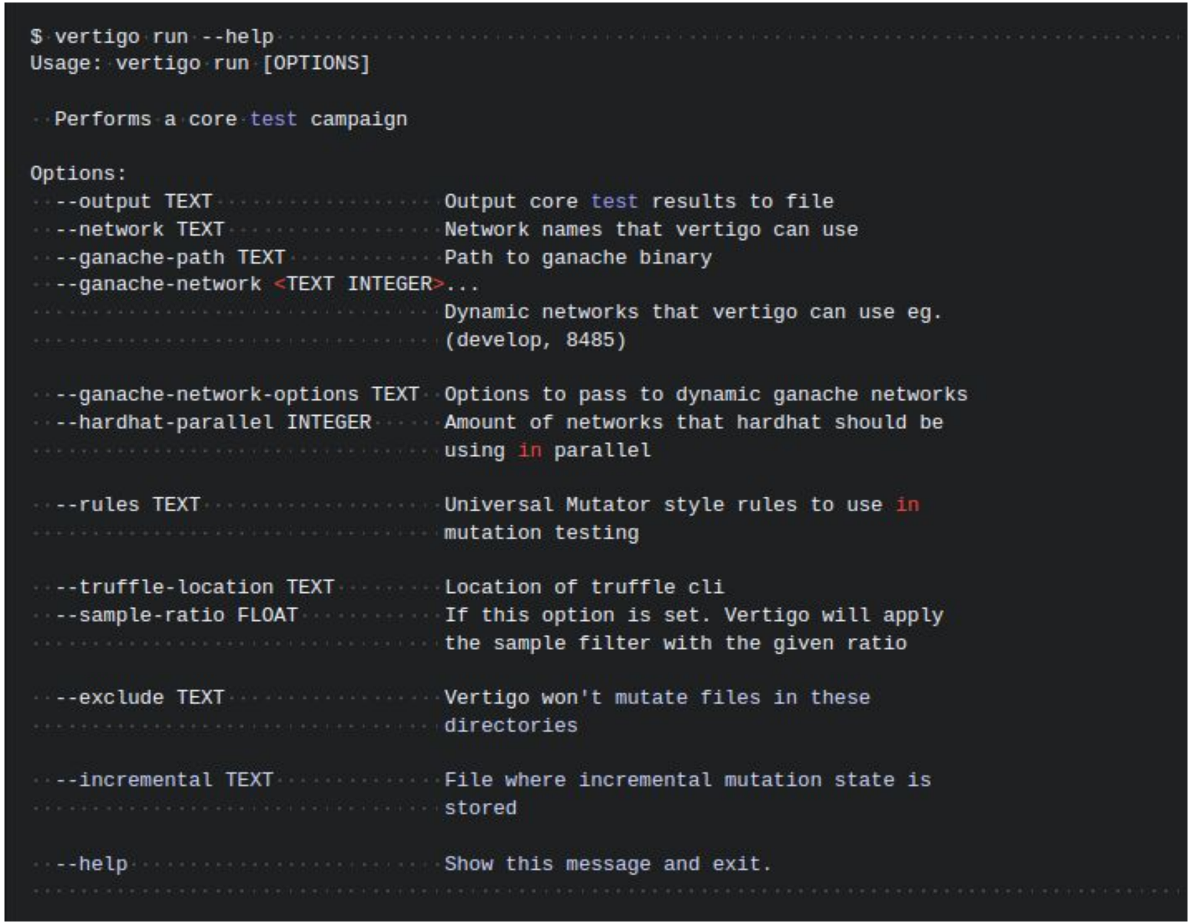
File		Mutation Test Score		Mt Killed Rate		Mesc Killed Rate		Compile Passed	
All File		63%	7/11	63%	7/11	100%	0/0	100%	11/11

File		Mutation Test Score		Mt Killed Rate		Mesc Killed Rate		Compile Passed	
Exchange.sol		75%	6/8	75%	6/8	100%	0/0	100%	8/8
AirSwapToken.sol		33%	1/3	33%	1/3	100%	0/0	100%	3/3

Mutation Testing Tools 2: Vertigo

https://github.com/JoranHonig/vertigo

Vertigo is a mutation testing framework designed to work specifically for smart contracts.
This mutation testing framework implements a range of mutation operators that are either selected from previous works or tailored to solidity.



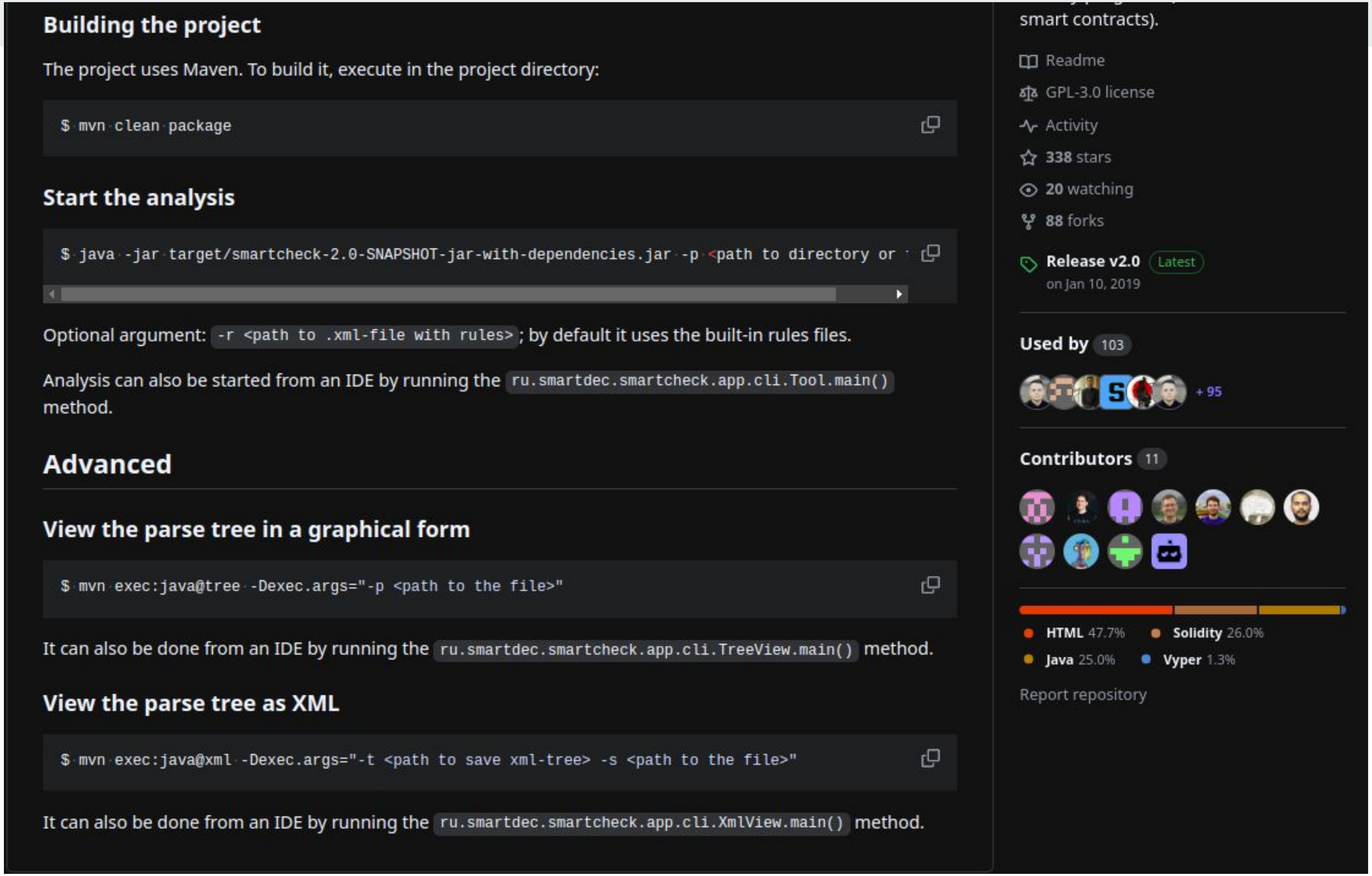
```
$ vertigo run --help
Usage: vertigo run [OPTIONS]

  Performs a core test campaign

Options:
  --output TEXT                   Output core test results to file
  --network TEXT                  Network names that vertigo can use
  --ganache-path TEXT             Path to ganache binary
  --ganache-network <TEXT INTEGER>...
                                  Dynamic networks that vertigo can use eg.
                                  (develop, 8485)

  --ganache-network-options TEXT  Options to pass to dynamic ganache networks
  --hardhat-parallel INTEGER      Amount of networks that hardhat should be
                                  using in parallel

  --rules TEXT                    Universal Mutator style rules to use in
                                  mutation testing

  --truffle-location TEXT         Location of truffle cli
  --sample-ratio FLOAT            If this option is set. Vertigo will apply
                                  the sample filter with the given ratio

  --exclude TEXT                  Vertigo won't mutate files in these
                                  directories

  --incremental TEXT              File where incremental mutation state is
                                  stored

  --help                          Show this message and exit.
```

Smart Contract Analysers: SmartCheck

https://github.com/smartdec/smartcheck

SmartCheck – a static analysis tool that detects vulnerabilities and bugs in Solidity programs (Ethereum-based smart contracts).

SmartCheck is a static evaluation tool for smart contracts. SmartCheck executes lexical and syntactical analysis on Solidity smart contract code. It generates intermediate representation as XML parse tree by using ANother Tool Language Recognition and a custom Solidity grammar. It detected suspicious samples by using XPath queries on the intermediate representation. The device attempted on various checked smart contracts referenced from Etherscan. Smart Check can recognize security, usefulness, operational and improvement issues. Examination shows that 99.9% of agreements have issues, 63.2% of agreements have basic weaknesses" provided by the author.



smart contracts).

Building the project

The project uses Maven. To build it, execute in the project directory:

```
$ mvn clean package
```

Start the analysis

```
$ java -jar target/smartcheck-2.0-SNAPSHOT-jar-with-dependencies.jar -p <path to directory or
```

Optional argument: `-r <path to .xml-file with rules>`; by default it uses the built-in rules files.

Analysis can also be started from an IDE by running the `ru.smartdec.smartcheck.app.cli.Tool.main()` method.

Advanced

View the parse tree in a graphical form

```
$ mvn exec:java@tree -Dexec.args="-p <path to the file>"
```

It can also be done from an IDE by running the `ru.smartdec.smartcheck.app.cli.TreeView.main()` method.

View the parse tree as XML

```
$ mvn exec:java@xml -Dexec.args="-t <path to save xml-tree> -s <path to the file>"
```

It can also be done from an IDE by running the `ru.smartdec.smartcheck.app.cli.XmlView.main()` method.

Readme
GPL-3.0 license
Activity
338 stars
20 watching
88 forks

Release v2.0 Latest
on Jan 10, 2019

Used by 103
+ 95

Contributors 11

● HTML 47.7% ● Solidity 26.0%
● Java 25.0% ● Vyper 1.3%

Report repository

Smart Contract Analysers: Securify

https://github.com/eth-sri/securify2

SECURIFY 2.0

new!

Challenges

- lack of best practices
- lack of specialized tools

Reference

Sujeetha, R., and CA S. Deiva Preetha. "A literature survey on smart contract testing and analysis for smart contract based blockchain application development." *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*. IEEE, 2021.