SuMo

A Mutation Testing Strategy for Solidity Smart Contracts

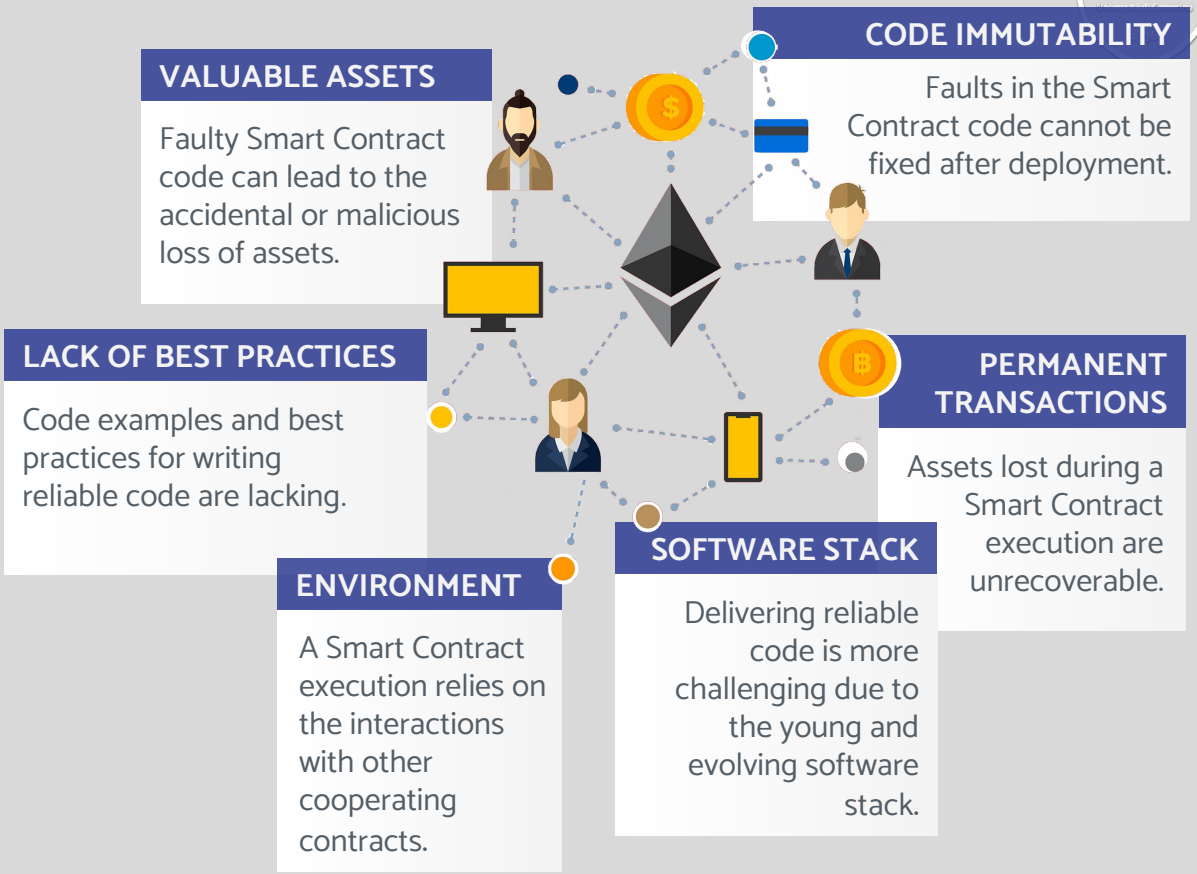
Morena Barboni Andrea Morichetta Andrea Polini

nome.cognome@unicam.it
School of Science and Technology
University of Camerino - ITALY

Workshop multidisciplinare su blockchain e DLT: incontro fra accademia e imprese
Firenze, 7-8 Novembre 2022

QUALITY ASSURANCE FOR SMART CONTRACTS
WHY SHOULD WE CARE?

- Smart Contracts must comply with high **reliability standards**.
- Developing and testing Smart Contracts present **unique challenges** tied to the blockchain environment.
- The developer community requires tools for **assessing the quality** of their **testing activities.**



| Mutation Testing

Mutation Testing is a powerful fault-based testing technique.

- Certain elements of the target program are mutated to mimic a typical programming fault
- The fault-injection process aims to:

EVALUATE THE ADEQUACY OF THE TEST SUITE IN FINDING REAL FAULTS

+

GUIDE THE IMPROVEMENT OF THE TEST DATA BASED ON UNDETECTED FAULTS

MUTATION TESTING
THE PROCESS

ORIGINAL PROGRAM P

Target of the Mutation Testing process

```
function update(uint price)
private returns(uint) {
  require(price > 0)
  p = price;
  return p;
}
```

MUTATION OPERATORS

Set of fault injection rules.

Each operator specifies how the code of P should be modified.

- BOR
- FVR
- RSD

MUTANTS OF P

Mutated versions of the original program.

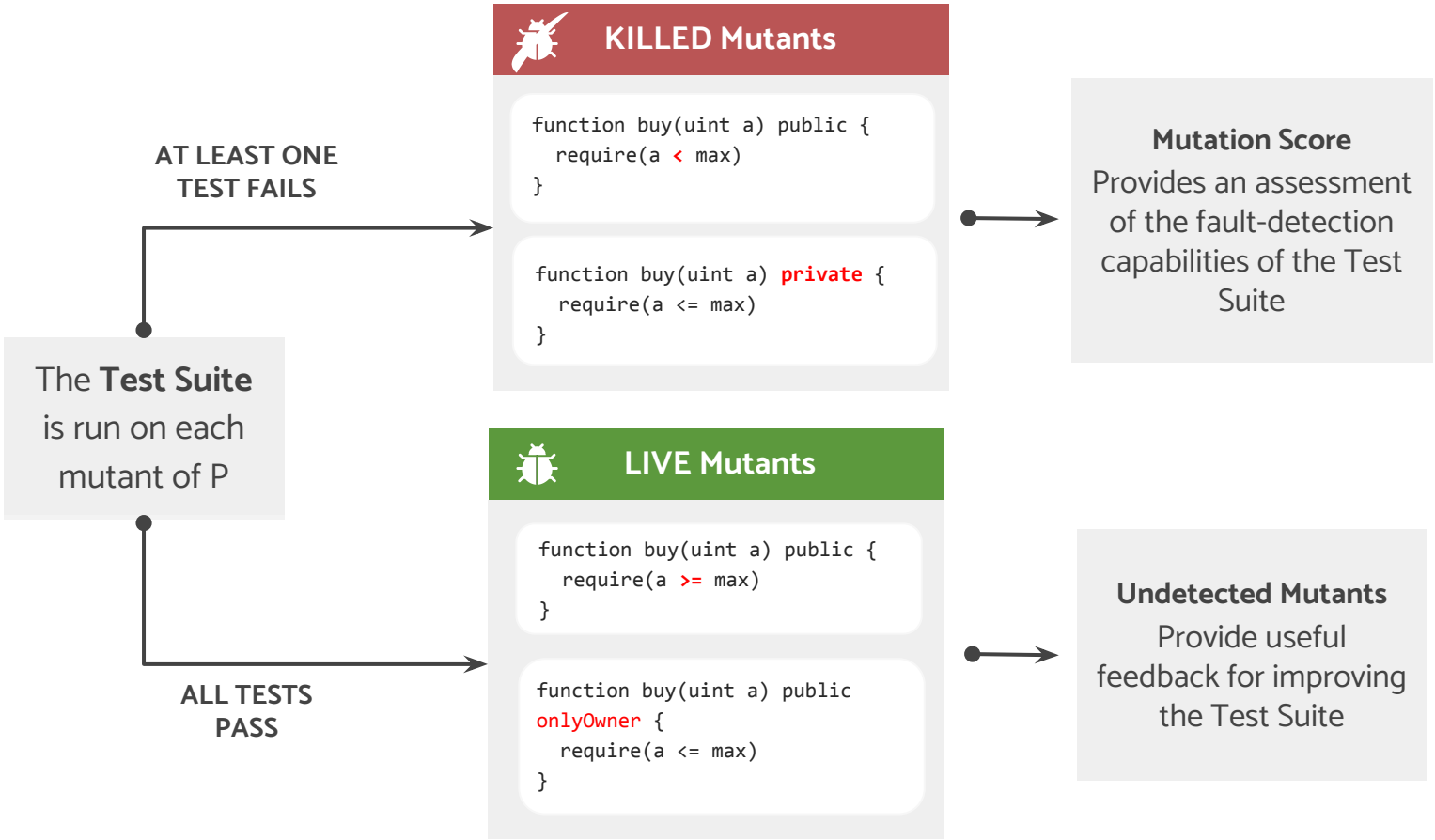
Each Mutant contains a minor change that mimics a common programming fault.

```
function update(uint price) private
returns(uint) {
  require(price >= 0)
  ...
}
```

```
function update(uint price) public
returns(uint) {  ...  }
```

```
function update(uint price) private
returns(uint) {
   ...
  // return p;
}
```

Mutation Testing
The Process



Cost Reduction

1 Limiting Stillborn Mutants

STILLBORN MUTANTS
are killed by the compiler

- ➔ **SELECTIVE MUTATION**
  We select operators that are empirically found to limit stillborn mutants.
- ➔ **MUTATION RULES**
  SuMo collects semantic information during the visit of the AST to improve the efficacy of the mutations.

2 Limiting Redundant Mutants

REDUNDANT MUTANTS
do not provide new information about the Test Suite quality

- ➔ **SELECTIVE MUTATION**
  We select operators that are empirically found to limit redundancy.
- ➔ **MUTATION RULES**
  Mutations that are likely to generate redundant mutants are merged.

3 Customized Mutation Process

Testers can select:

- ➔ Mutation Operators
- ➔ Contract Files

SuMo Mutation Operators

Type	Class	Operator ID	Class	Operator ID
Solidity Specific	Address	AVR, SCEC	Functions	PKD, RSD, RVS
	Block and Transaction Properties	GVR, TOR	Function Modifiers	MOC, MOD, MOI, MOR, OMD
	Constructor	CCD	Global Functions	MCR, SFD, SFI
	Data Location	DLR	Libraries	SFR
	Ether Transfer	ETR	Operators	DOD
	Events	EED	Units	VUR
	Exception Handling	EHC	Visibility	FVR, VVR
General	Control	BCRD, CBD, CSC, LSC	Overriding	ORFD, SKD, SKI
	Expressions	AOR, BOR, ICM, UORD	Overloading	ACM, OLFD
	Literals	BLR, HLR, SLR	Types	ECS, ER

SuMo currently implements 44 Mutation Operators

Implementation

TECHNOLOGIES

NodeJS

Lightweight Javascript run-time environment

solidity-parser-antlr

Solidity parser for Javascript built on top of a robust antlr4 grammar

SuMo

Truffle

Popular testing framework for Ethereum Smart Contracts

Ganache

Sets up a local Ethereum blockchain for deploying and testing Smart Contracts

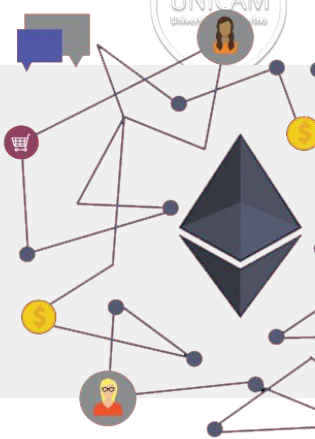
https://github.com/MorenaBarboni/SuMo-SOlidity-MUtator

Validation

Case Studies

2 Open-Source Ethereum dApps

DApp		
Name	Smart Contracts	LOC
ETHER CROWDFUNDING	Crowdfunding Campaign.sol	428
BIONIC EVENT DAPP	Event.sol EventFactory.sol	182



DApp	Test Suite			
	Size	LOC	Stmt. Coverage	Branch Coverage
ETHER CROWDFUNDING	35	902	93.83	71.3
BIONIC EVENT DAPP	25	261	90	65

Shipped with high-coverage Test Suites

Experimental Results (1/2)

Main Findings

1 The selected applications achieve **low Mutation Scores.**

Test Suites with good coverage values do not ensure code reliability.

2 The Mutation Score is particularly low for Solidity-specific mutations

Addressing the distinctive mechanisms of the Solidity language is particularly challenging for Smart Contract developers.

3 SuMo generates a relatively low number of stillborn mutants (~ 10%)

Application	Mutation Operators	Mutants		Mutation Score (%)
		Tot.	Stillborn	
ETHER CROWDFUNDING	ALL	681	59	**47,7**
	SOLIDITY	401	38	**28,9**
BIONIC EVENT DAPP	ALL	189	29	**58,7**
	SOLIDITY	148	29	**54,7**

TABLE V: Experimental Results

Experimental Results (2/2)

Operators	Mutants			MS (%)
	Tot.	Stillborn	Killed	
ALL	189	29	74	**58,7**
SOLIDITY	148	29	52	**54,7**

BIONIC-EVENT-DAPP

Operators	Mutants			MS (%)
	Tot.	Stillborn	Killed	
ALL	172	29	114	**86,8**
SOLIDITY	136	29	87	**85**

BIONIC-EVENT-DAPP
(After the Mutation Analysis)

Analysing the live mutants allowed us to:

- Improve the existing test data and design additional test cases;
- Identify and correct issues in the SUT (Software Under Test);
- Improve the fault-detection capabilities of the provided Test Suite.

Conclusions

- **Mutation Score** is a more reliable indicator of the Test Suite quality
  - Mutation Testing can benefit business-critical programs like Smart Contracts.
- The preliminary validation of **SuMo** provided encouraging results:
  - The faults injected by SuMo were frequently missed by real Test Suites;
  - Mutation analysis is a feasible approach for improving the Mutation Score.
- **SuMo** can help developers to:
  - write higher quality Test Suites
  - deliver more reliable Solidity applications.

Thank you for your attention!