

CS396A (UGP-II) Report

Name - Shivamshree Gupta

Supervisor - Subhajit Roy

Title: A Decentralized Application for Testing and Identifying Smart Contract Vulnerabilities

Introduction

With its potential for decentralisation, transparency, and security, blockchain technology has revolutionised a number of industries. The idea of smart contracts is one of the major advancements brought forth by blockchain technology. The terms of the agreement between the parties are explicitly specified in lines of code for these self-executing contracts. Smart contracts have become more popular, but this has also made them more vulnerable to risks like bugs and vulnerabilities, which can result in lost money and reputational harm. This report introduces a decentralised application (DApp) made to automatically find smart contract security flaws, hence enhancing their overall security.

The Evolution of the Project

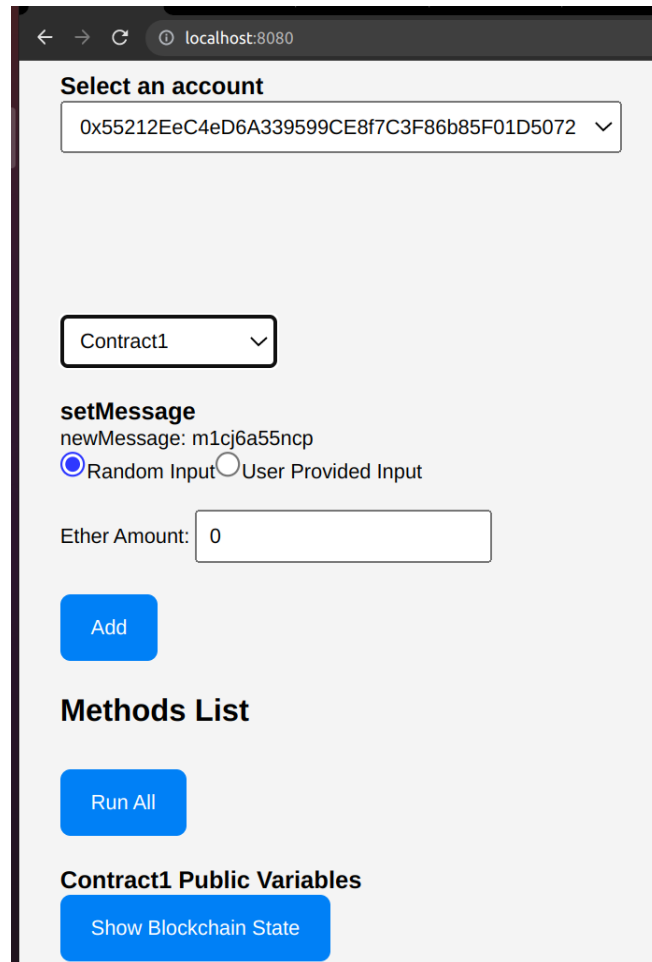
The project began as a DApp that interacted with smart contracts and stored data on the blockchain. It then evolved into an integrated smart contract testing solution. By leveraging the fuzzer-testing analogy, the DApp allows users to test public methods and identify vulnerability patterns in smart contracts.

Project Overview

The project aims to automate the process of detecting vulnerabilities in smart contracts, improving overall security. Our tool analyses different smart contracts deployed on a blockchain by allowing user to execute public methods using various inputs to detect vulnerability patterns. Our project provides an easy-to-use solution to help developers identify and fix potential issues before they can be exploited.

So, in the following screenshot of *Contract Explorer*, first we need to connect to some *account* on the blockchain network, and for that the project used *Metamask*. After that, there is a dropdown that shows all the *contracts deployed*. Selecting one contract will display all its *public methods* along with the *input parameters* it is supposed to take. There is a *method list* maintained where we can add as many public

methods along with random or user defined input and also choose *ether values* for transactions in case of payable contracts. Methods List consists of ***Run All***, which runs all the public methods present in the list one by one. ***Show Blockchain State*** will show the public variable's information of the selected contract.



The screenshot shows a web application running on localhost:8080. The interface includes a 'Select an account' dropdown menu with the address '0x55212EeC4eD6A339599CE8f7C3F86b85F01D5072' selected. Below this is a 'Contract1' dropdown menu. The 'setMessage' section shows 'newMessage: m1cj6a55ncp' and two radio buttons: 'Random Input' (selected) and 'User Provided Input'. An 'Ether Amount' input field is set to '0'. There is an 'Add' button. The 'Methods List' section has a 'Run All' button. At the bottom, the 'Contract1 Public Variables' section has a 'Show Blockchain State' button.

Tech Stack and Tools

The project is built using JavaScript, React, and Solidity, which are widely used languages and frameworks in the blockchain development community. The project relies on Web3.js for interacting with Ethereum-based smart contracts and Truffle for compiling, testing, and deploying contracts. A local-blockchain instance of Ganache is used for testing purposes.

Re-Entrancy Attack: A Common Vulnerability

A malicious contract continually calling a vulnerable contract's public function before the vulnerable contract has finished processing its previous function call is known as a re-entrancy attack. This may result in the malicious contract depleting the balance of the susceptible contract. The DAO breach in 2016 is a real-world illustration of this, when \$50 million worth of Ether was lost as a result of a re-entrancy vulnerability in its smart contract.

Identifying a Re-Entrancy Vulnerability

Let's showcase the detection of a re-entrancy vulnerability in the EtherStore smart contract:

EtherStore Smart Contract

EtherStore is a smart contract that allows users to deposit and withdraw Ether. The contract contains a re-entrancy vulnerability in its withdrawal function, making it susceptible to attacks.

Select an account

0xC489fD3bDE48A97a6Efa428205cA1651405cA1f0

EtherStore

deposit

☒ Random Input ☐ User Provided Input

Ether Amount: 3

Add

withdraw

☒ Random Input ☐ User Provided Input

Ether Amount: 0

Add

Methods List

deposit

Delete

Inputs:

Run All

EtherStore Public Variables

Show Blockchain State

- balances (0xC489fD3bDE48A97a6Efa428205cA1651405cA1f0): 3000000000000000000
- getBalance: 3000000000000000000

So, say via above account I deposit 3 ethers to etherstore, then balances will be 3 ethers and getBalance (total balance of the etherstore contract) will also return 3 ethers.

The Attack Smart Contract

The attack contract is designed to exploit the re-entrancy vulnerability present in the EtherStore smart contract. It uses a **fallback function** and a **receive function** to recursively call the EtherStore's **withdraw** function, allowing it to drain the EtherStore's balance. The attack is initiated by calling the **attack** function, which deposits Ether into the EtherStore and triggers the recursive withdrawal.

[illegible]

Now, if I call attack method with 1 ether, then the attack contract will first deposit 1 ether and then withdraw the complete etherstore balance which is 4 ethers and getBalance of attack contract will then return 4 ethers as shown.

Identify the Vulnerability

Users can simulate the re-entrancy attack by executing the "attack" function of the Attack contract, which will trigger the recursive calls to EtherStore's **withdraw** function. Users can monitor the state of the EtherStore contract by checking its balance and individual account balances using the **Show Blockchain State** button

Conclusion

Developers may quickly find and address potential problems in smart contracts using the DApp for vulnerability detection before they can be exploited. Smart contracts and the sectors they serve will be more secure in the future thanks to the DApp's automated vulnerability detection method, which also helps to increase the trust and reliability of blockchain-based apps.