A Decentralised Application for Testing and Identifying Smart Contract Vulnerabilities

CS396A - Undergraduate Project

Name : Shivamshree Gupta

Supervisor : Subhajit Roy

The Rise of Blockchain Technology

1 What is Blockchain Technology?

Decentralized, transparent, secure ledger system that records transactions using cryptography, eliminating the need for intermediaries, enabling trust, and revolutionizing industries with its potential for innovation.

2 Advantages of Blockchain Technology

Decentralization, transparency, security, trust, elimination of intermediaries, increased efficiency, cost reduction, potential for innovation in various industries.

3 Applications of Blockchain Technology

Finance (cryptocurrency), supply chain , healthcare (patient records, clinical trials), and more.

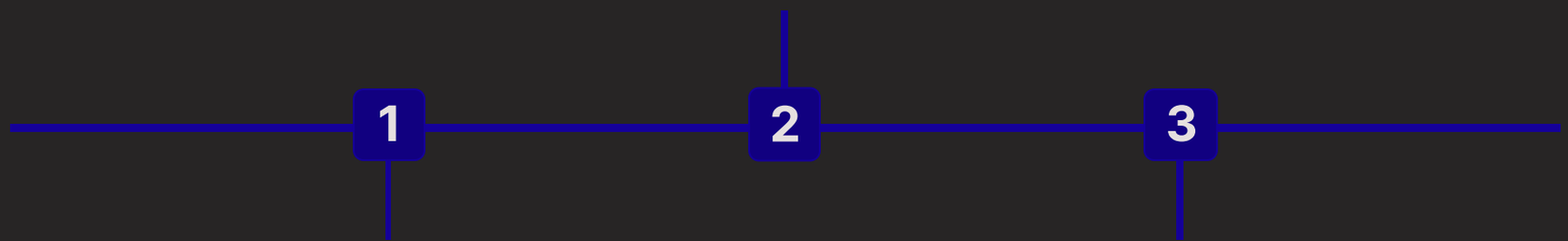
4 The Future of Blockchain Technology

Promising future, revolutionizing value and information exchange.

The Importance of Smart Contract Security

The Risks of Smart Contracts

Smart contracts are vulnerable to bugs and vulnerabilities, which can lead to financial loss, reputation damage, and legal liability. Hackers can exploit bugs to steal funds, manipulate data, or disrupt the network.



What are Smart Contracts?

Smart contracts are self-executing contracts with the terms of the agreement between buyer and seller being directly written into lines of code.

The Importance of Smart Contract Security

Securing smart contracts is crucial to ensuring the trust and reliability of blockchain-based applications.

From Dapp to Smart Contract Testing: The Evolution of Our Project

1

Phase 1: Dapp Development

Our project began as a Dapp that interacted with smart contracts and stored data on the blockchain.

2

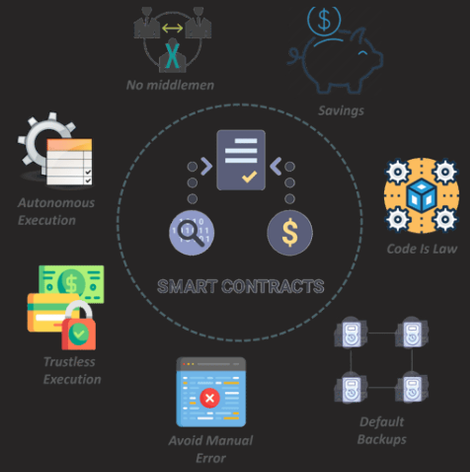
Phase 2: Smart Contract Testing

We integrated smart contract testing into the project, using the fuzzer-testing analogy to create a way for users to test public methods and identify vulnerability patterns.

Project Overview: DApp for Vulnerability Detection

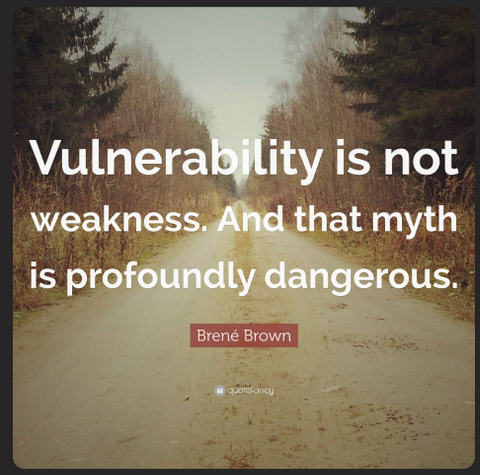
What is Our Project?

The project aims to automate the process of detecting vulnerabilities in smart contracts, improving overall security.



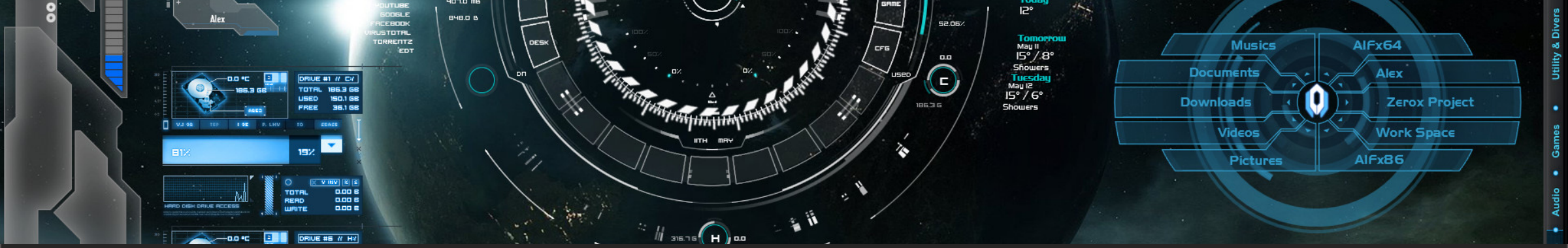
How Does It Work?

Our tool analyses different smart contracts deployed on a blockchain by allowing user to execute public methods using various inputs to detect vulnerability patterns.



The Benefits of Our Project

Our project provides an easy-to-use solution to help developers identify and fix potential issues before they can be exploited.



Tech Stack and Tools

1 Programming Languages

The project is built using JavaScript, React, and Solidity, which are widely used languages and frameworks in the blockchain development community.

2 Tools and Libraries

The project relies on Web3.js for interacting with Ethereum-based smart contracts and Truffle for compiling, testing, and deploying contracts.

Re-Entrancy Attack: A Common Vulnerability

What is a Re-Entrancy Attack?

A re-entrancy attack occurs when a malicious contract repeatedly calls a vulnerable contract's public function before the vulnerable contract can finish executing its previous function call, causing the malicious contract to drain the vulnerable contract's balance.

Example

Real-world example: The DAO hack: In 2016, the DAO, a decentralized autonomous organization built on Ethereum, suffered a major attack resulting in a loss of around $50 million worth of Ether due to a re-entrancy vulnerability in its smart contract.

Application Demo: Identifying a Re-Entrancy Vulnerability

Step 1: EtherStore

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.

Step 2: 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.

Step 3: 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.

Benefits and Advantages of the Project

Improved Security

Our project provides an easy and accessible way to improve the security of smart contracts, reducing the risks of financial loss, reputation damage, and legal liability. The project's capabilities can be expanded to detect a wider range of vulnerabilities in smart contracts, providing more comprehensive security coverage for developers and organizations.

Cost-Effectiveness

Our project is cost-effective, as it reduces the need for manual auditing