AN ANALYSIS OF ESBMC'S EFFECTIVENESS ON SWC VULNERABILITIES

A REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF BACHELOR OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2023

John Lindon Robinson

Department of Computer Science

Contents

Word Count: 1431

List of Tables

List of Figures

Abstract

AN ANALYSIS OF ESBMC'S EFFECTIVENESS
ON SWC VULNERABILITIES
John Lindon Robinson
A report submitted to The University of Manchester
for the degree of Bachelor of Science, 2023

This university project aims to develop a benchmark for the ESBMC smart contract checker by creating a large set of vulnerable smart contracts. The smart contracts were written with Solidity, a popular programming language for creating smart contracts on the Ethereum blockchain.

The project involved researching common vulnerabilities in smart contracts and implementing them intentionally in the benchmark contracts. The vulnerabilities targeted include reentrancy attacks, integer overflows and underflows, logic flaws, and other common mistakes that can lead to security vulnerabilities.

The benchmark contracts were then tested with ESBMC, a popular software model checker that is capable of verifying the correctness of smart contracts. The results of the tests were analyzed to determine the effectiveness of ESBMC in detecting and preventing vulnerabilities in smart contracts.

The project aims to provide a useful resource for developers and researchers working on smart contract security. By providing a comprehensive set of vulnerable smart contracts, the benchmark can be used to test and compare different smart contract security tools and techniques, ultimately leading to more secure smart contracts and blockchain applications.

Declaration

No portion of the work referred to in this report has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank... TODO:

Chapter 1

Introduction

1.1 Motivation

The growing popularity and widespread adoption of smart contracts in various industries have sparked significant interest in ensuring their security, reliability, and robustness. Smart contracts, as self-executing contracts with the terms of the agreement directly written into code, are becoming an essential component of blockchain platforms, such as Bitcoin, Ethereum, Polygon or a suite of other Platforms. They enable secure, decentralized, and automated transactions without intermediaries, increasing efficiency and reducing transaction costs. Despite these potential advantages, poorly designed or vulnerable smart contracts can lead to significant losses, as evidenced by high-profile security breaches, such as a hack on The DAO in 2016, which cost its users 50 Million USD, by exploiting a particular vulnerability.

This dissertation aims to study the effectiveness of the ESBMC (Efficient SMT-Based Context-Bounded Model Checker) in identifying vulnerabilities within smart contracts. ESBMC is a powerful and widely-used tool for formal verification, which has shown great potential in detecting software errors, including concurrency-related bugs and vulnerabilities in smart contracts. However, there is a shortage of research on the performance of ESBMC when applied to smart contracts with known vulnerabilities.

By developing a set of intentionally vulnerable smart contracts to serve as a benchmark, this study seeks to bridge this research gap and provide valuable insights into the capabilities and limitations of ESBMC in detecting and mitigating smart contract vulnerabilities. This benchmark suite will not only help assess the efficiency and accuracy of the ESBMC tool but also contribute to improving and enhancing model-checking

techniques for smart contract verification.

Additionally, the findings of this research can assist smart contract developers and security experts in gaining a deeper understanding of potential vulnerabilities and employing more secure coding practices. By evaluating the ESBMC tool's performance on this benchmark, the study also hopes to foster the development of more robust and effective model-checking tools for smart contract verification, ultimately establishing safer and more trustworthy blockchain ecosystems.

1.2 Research Question, Aim and Objectives

1.2.1 Research Question

How effectively is ESBMC detecting and mitigating vulnerabilities within a benchmark suite of intentionally vulnerable smart contracts?

1.2.2 Aim

This study aims to assess the performance of ESBMC in identifying and addressing vulnerabilities in smart contracts found in the SWC Registry [3] by using a benchmark suite of deliberately vulnerable smart contracts.

1.2.3 Objectives

The objectives for this project are as follows:

1. To develop a benchmark suite of vulnerable smart contracts that simulate real-world security flaws and weaknesses.

2. To evaluate the effectiveness of ESBMC in detecting and analyzing the vulnerabilities within the benchmark suite.

3. To investigate the limitations and challenges of using ESBMC for smart contract verification.

4. To provide recommendations for improving the performance of ESBMC and other model-checking tools in detecting and mitigating smart contract vulnerabilities.

Chapter 2

Background

2.1 Blockchain

A blockchain is a decentralized, distributed ledger technology that enables secure, transparent, and tamper-resistant storage of digital records across a network of participants. It consists of a series of blocks, each containing a list of transactions, which are cryptographically linked and secured using cryptographic algorithms. This structure allows for enhanced security and data integrity, as altering the information in one block would require most network participants' consensus and modifying all subsequent blocks.

Although initially developed for supporting cryptocurrencies like Bitcoin, blockchain technology has evolved to accommodate various applications across many industries, such as finance, supply chain, healthcare, and more [4] [2]. Ethereum has emerged as a leading platform for developing and deploying smart contracts among the different blockchain platforms.

2.1.1 Etherum Blockchain

The Ethereum blockchain, launched in 2015 by Vitalik Buterin and his team, was designed to facilitate the creation, management, and execution of decentralized applications (DApps) and smart contracts. Unlike Bitcoin, which is primarily used for transferring digital currency, Ethereum provides a decentralized virtual machine—the Ethereum Virtual Machine (EVM)—which can execute arbitrary Turing-complete code on the blockchain [1]. This feature allows developers to build and deploy more complex and versatile applications on the Ethereum platform.

Smart contracts are self-executing contracts with the terms of the agreement directly written into code. They automatically execute and enforce the contract's terms when predefined conditions are met without the need for intermediaries. This enables secure, decentralized, and automated transactions on the blockchain, leading to increased efficiency and reduced transaction costs. Ethereum's native cryptocurrency, Ether (ETH), is used to pay for the computational resources and transaction fees required to execute smart contracts on the network. A pertinent fact in this research is that once a smart contract is deployed on the Ethereum blockchain, it cannot be modified or removed. This immutability makes smart contracts a highly attractive target for attackers, as they can potentially cause significant financial losses and damage the organization's reputation. For this reason, smart contracts must be developed and deployed securely.

2.1.2   The Solidity Language

Solidity is a high-level, statically-typed, contract-oriented programming language specifically designed for writing smart contracts on the Ethereum blockchain. Created by Dr. Gavin Wood, Christian Reitwiessner, and their team at Ethereum, Solidity is influenced by other programming languages such as JavaScript, Python, and C++, and is designed to target the Ethereum Virtual Machine (EVM). The EVM executes the compiled bytecode of the smart contracts, which is generated from the Solidity source code.

2.1.2.1   Syntax and Structure

Solidity syntax is similar to JavaScript and employs a curly-bracket () notation for defining code blocks. A Solidity smart contract typically starts with a *pragma* directive, which specifies the version of the Solidity compiler required for the source code. This is followed by the contract definition, which includes the contract's state variables, functions, events, and access modifiers.

```
pragma solidity ^0.8.0;

contract SimpleStorage {
    uint256 private storedData;
```

```
function set(uint256 x) public {
    storedData = x;
}

function get() public view returns (uint256) {
    return storedData;
}

}
```

The example above demonstrates a simple Solidity contract, *SimpleStorage*, which allows users to store and retrieve an unsigned 256-bit integer value. The contract consists of a private state variable, *storedData*, and two public functions, *set()* and *get()*.

2.1.2.2 Data Types and Variables

Solidity supports various data types, including value types (such as integers, booleans, and addresses) and reference types (such as arrays, mappings, and structs). Additionally, Solidity allows for the declaration of user-defined types, such as enums and structs, to create more complex data structures.

2.1.2.3 Functions and Modifiers

Functions in Solidity are similar to functions in other programming languages, defining a reusable block of code that performs a specific task. Functions can be declared as public, private, external, or internal, which determines their visibility and accessibility within the contract and by other contracts. Functions can also be marked as *view* or *pure*, indicating that they do not modify the contract's state and only read or compute data, respectively.

Modifiers can be used to alter the behavior of functions by appending or prepending additional code to the function's body. They are often used to enforce access control, by requiring certain conditions to be met before the function can be executed, such as requiring the sender to be the contract owner.

2.1.2.4   Events and Inheritance

Events are used in Solidity to emit logs that can be monitored by the contract's users, allowing them to be notified of specific occurrences or state changes within the contract. This is particularly useful for creating event-driven applications and tracking transactions on the Ethereum blockchain.

Solidity also supports inheritance, allowing contracts to inherit properties and methods from other contracts. This enables code reuse and modularity, facilitating the development of complex and robust smart contracts.

By understanding the fundamentals of Solidity and its features, developers can create secure and efficient smart contracts on the Ethereum platform. The background knowledge on Solidity provided in this subsection serves as a foundation for the subsequent analysis of smart contract vulnerabilities and the evaluation of ESBMC in this study.

Bibliography

[1] E. Hildenbrandt, M. Saxena, X. Zhu, N. Rodrigues, P. Daian, D. Guth, and G. Roşu. Kevm: A complete semantics of the ethereum virtual machine. 2017.

[2] IBM. Blockchain for financial services. `https://www.ibm.com/blockchain/industries/financial-services`.

[3] MythX. Swc registry. `https://swcregistry.io/`, 2020.

[4] M. Xu, X. Chen, and G. Kou. A systematic review of blockchain. *Financial Innovation*, 5(1), 2019.

Appendix A

Appendix Title

A.1 Example First Section

TODO:

A.2 Output

TODO: