DOCTORAL THESIS

Pre-deployment Analysis of Smart Contracts

Sundas Munir

Halmstad University | School of Information Technology
Doctoral Thesis | Halmstad University Dissertations no. 120

Pre-deployment Analysis of Smart Contracts

Sundas Munir

DOCTORAL THESIS | Halmstad University Dissertations no. 120

Pre-deployment Analysis of Smart Contracts

Halmstad University Dissertations no. 120
ISBN 978-91-89587-57-1 (printed)
ISBN 978-91-89587-56-4 (pdf)
Publisher: Halmstad University Press, 2024 | www.hh.se/hup
Printer: Media-Tryck, Lund

Abstract

Smart contracts are programs that reside and execute on top of blockchains. These programs commonly perform financial transactions and contain the back-end logic of various blockchain-supported applications. The presence of errors and bugs in smart contracts poses security risks to the applications they support. This is especially concerning because operations performed by smart contracts are irreversible by design, which is a key feature enforced by blockchain technology. Thus, ensuring the correctness and security of smart contracts before deployment is crucial. To achieve this, several program analysis and verification approaches are being actively researched and applied to smart contracts.

The volume of research in this area makes it challenging to articulate the state-of-the-art. The first contribution of this thesis is an investigation into how pre-deployment analysis techniques have been applied to ensure the correctness and security of smart contracts. This investigation factors out the relationship between vulnerabilities in smart contracts and pre-deployment analysis techniques through properties they address.

Among the issues uncovered by the investigation, one notable set pertains to nondeterministic (ND) factors involved in smart contract execution in the Ethereum blockchain. For example, transactions (function invocations) dispatched to Ethereum smart contracts are scheduled in ND order, and inputs received during execution, such as inputs from asynchronous callbacks of Oracles and externally called contracts that halt unexpectedly, are nondeterministic. Consequently, these factors may cause risks of ND changes to the state of smart contracts. The second contribution of this thesis is the proposal of methods for the static (at pre-deployment) detection of program sites (for example, state variables or cryptocurrency transfer instructions) susceptible to ND changes due to ND transaction scheduling or ND inputs and return values (from asynchronous callbacks and externally called contracts) in Ethereum smart contracts. The evaluations show that our approaches can outperform existing methods with respect to efficacy and runtime.

To my Sanam.

Acknowledgments

I wish to express my heartfelt appreciation to my principal supervisor, Walid Taha, for guiding and encouraging me in defining my research scope and achieving my goals, even when the road got tough. Throughout this entire journey, Walid's continuous support, assistance, and motivation have been invaluable to me.

I am profoundly grateful to my co-supervisors, Wojciech Mostowski and Mark Dougherty, for their invaluable contributions to my research. I also had the pleasure of collaborating with Christoph Reichenbach, whose constructive feedback significantly improved the quality of my work. Their support has been instrumental throughout my research journey, and I sincerely appreciate their guidance and expertise.

My sincere gratitude goes to all the ITE members of Halmstad University for their unwavering support throughout my studies.

My deepest thanks goes to my husband, whose unwavering love and steadfast support have been my anchor, helping me navigate life's most challenging moments. This achievement would not have been possible without his constant support and encouragement.

Last but not least, my heartfelt appreciation goes to my parents and brothers. To my loving mother, thank you for your boundless love and encouragement; it means the world to me. And to my father, whose passing has left a deep void in my heart—I wish he were here to see this moment, as every success I achieve is a tribute to his sacrifices and unwavering support. His memory is a light in my life, guiding me forward every day.

List of Papers

The following papers, referred to in the text by their Roman numerals, are included in this thesis.

PAPER I: **Pre-deployment Analysis of Smart Contracts - A Survey**
Sundas Munir, Walid Taha. *Manuscript. arXiv:2301.06079*

PAPER II: **TODler: A Transaction Ordering Dependency anaLyzER - for Ethereum Smart Contracts**
Sundas Munir, Christoph Reichenbach. *2023 IEEE/ACM 6th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), Melbourne, Australia, 2023, doi: 10.1109/WETSEB59161.2023.00007.*

PAPER III: **Statically Checking Transaction Ordering Dependency in Ethereum Smart Contracts**
Sundas Munir, Walid Taha. *The 6th ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI:'24), 2024, Singapore, Singapore. ACM, New York, NY, USA, https://doi.org/10.1145/3659463.3660013*.

PAPER IV: **Statically Checking Missing Input Validations in Solidity Smart Contracts - A Case Study**
Sundas Munir, Mirza Sanam Iqbal Baig, Mah Noor, Syeda Hina Murad. *2023 IEEE International Conference on Blockchain (Blockchain), Danzhou, China, 2023, doi: 10.1109/Blockchain60715.2023.00017.*

PAPER V: **Static Detection of Missing Validations in Solidity Smart Contracts**
Sundas Munir, Walid Taha, Mirza Sanam Iqbal Baig. *2024 IEEE International Conference on Cyber Security and Resilience (CSR), London, United Kingdom, 2024, doi: 10.1109/CSR61664.2024.10679381*

Contents

Abbreviations

DoS	Denial-of-Service
ETH	Ether
EVM	Ethereum Virtual Machine
FONOVA	Flow Of Non-deterministic Values Analyzer
MEV	Maximal Extractable Value
MIV	Missing Input Validation
MIV-Checker	Missing Input Validation Checker
MRV	Missing Return values Validation
ND	Nondeterministic
NFT	Non-Fungible Token
RAW	Read-After-Write
TA	Transaction ordering dependency on Amount
TD	Transaction ordering dependency on execution of `delegatecall`
TOD	Transaction Ordering Dependency
TODChecker	Transaction Ordering Dependency Checker
TODler	Transaction Ordering Dependency analyzer
TR	Transaction ordering dependency on Receiver
TRS	Transaction ordering dependency on Receiver of `selfdestruct`
TS	Transaction ordering dependency on execution of `selfdestruct`
TST	Transaction ordering dependency on the Sender of Token
TT	Transaction ordering dependency on Transfer

1. Introduction

Ethereum is a prominent blockchain platform that facilitates the execution of stateful smart contracts written in high-level languages like Solidity or Vyper. Being computer programs, smart contracts can be susceptible to programming bugs and logical flaws, which can create security vulnerabilities. These vulnerabilities often result in significant financial losses, as smart contracts typically handle transactions involving valuable digital assets such as cryptocurrencies or crypto-tokens, and support decentralized applications like decentralized exchanges and social networks. Furthermore, smart contract transactions are irreversible by design, a feature enforced by blockchain technology, making fixing vulnerabilities in smart contracts after deployment challenging. This necessitates rigorous security measures to prevent vulnerabilities that could lead to significant financial or operational losses.

1.1 Challenges and Motivation

Traditional program analysis and verification methods have been used to ensure the security and correctness of smart contracts before deployment [1–3]. Work in this area has introduced a large and diverse number of analysis methods to address numerous smart contract vulnerabilities. This necessitates a comprehensive assessment of the smart contract vulnerabilities that have been mitigated and the specific methods employed. However, simply linking the vulnerabilities to their detection methods has two drawbacks. Firstly, there are important relationships between different vulnerabilities that require additional structure to expose. Secondly, without such structure, the seeming diversity in methods and vulnerabilities makes such a connection sparse and uninformative. To address this challenge, I connect vulnerabilities and methods via properties. Intuitively, a program property is any statement about the syntax or behavior of a given program. Examples of properties include depositing only positive values or always selecting the highest bid among a set of offers. If a smart contract does not satisfy a specific property, it could indicate one or more potential vulnerabilities. Moreover, analysis methods often use program properties to fully or partially specify the correctness of a contract. Finally, program properties can help classify smart contract vulnerabilities and

link them to their respective detection methods.
As for properties, smart contracts can be required to satisfy certain properties for secure and reliable execution. For instance, we may wish to require them to execute deterministically. Deterministic execution means that a smart contract will always produce the same output given the same inputs and initial state. For Ethereum smart contracts, nondeterminism is not intended and creates an attack surface allowing potential attackers to exploit the inconsistencies and manipulate the contract's behavior in unintended ways. The literature identifies certain sources of nondeterminism within the Ethereum ecosystem that can lead to risks of unexpected state changes in smart contracts [4–6]. Three of such sources are as follows:

1. Ethereum nodes, called validators or miners, schedule a set of transactions in the next/upcoming blocks in arbitrary order, resulting in nondeterministic (ND) scheduling of transactions. Even though transactions scheduled within a block are executed sequentially and atomically, nondeterministic scheduling (ND-scheduling) can introduce the risk of unexpected state changes in smart contracts due to their shared state that transactions can access and modify [4; 5].

2. Oracle services furnish external data, like weather forecasts or stock prices, to smart contracts through asynchronous callbacks. These callbacks return in an ND order, which causes inputs from Oracles to potentially arrive after a contract's state has been modified, resulting in read-after-write (RAW) hazards or inconsistent state changes [4–8]. Additionally, inputs from users and other contracts may cause unexpected or unpredictable changes to a contract's state during its execution. For example, loops whose iterations depend on user input may unpredictably become too expensive to execute [4]. Consequently, because user inputs and Oracle responses can introduce ND behavior into a contract's execution, these inputs have been considered a form of language-level nondeterminism in the context of Ethereum smart contract execution [4].

3. Smart contracts often include calls to other smart contracts, which may behave inconsistently. For example, an externally called contract may not execute completely; as such, it may require more execution cost (gas) than provided with the transaction. Thus, the return values from external calls and crypto-asset transfers will differ depending on whether the execution of the external contract succeeds, making these values nondeterministic [4].

ND-scheduling causes vulnerabilities when a prior transaction modifies a contract's state and a subsequent transaction executes critical operations, such as

cryptocurrency transfers, based on that state. As a result, a dependency is formed, commonly known as Transaction Ordering Dependency (TOD) [1; 2; 9–12]. Existing studies [7; 9; 13; 14] addressing TOD-related vulnerabilities suffer from various limitations, such as a high number of false positives or false negatives, limited detection of TOD vulnerabilities, and the inability to identify TOD issues arising from multiple interacting contracts.
Furthermore, failing to validate inputs from users, other contracts, or asynchronous callbacks, as well as return values from users or other contracts, can expose critical program parts—such as cryptocurrency transfers or smart contract termination instructions—to untrusted and unpredictable external inputs [15]. When such inputs or return values are not validated using conditions (to meet some preconditions), we have unvalidated values. We refer to the issue of unvalidated inputs being used in critical program parts as Missing Input Validation (MIV) and the issue of unvalidated return values being used in critical program parts as Missing Return Value Validations (MRV). Existing approaches [13; 15; 16] to detect MIV and MRV suffer from various limitations, including a high number of false negatives. We refer to TOD, MIV, and MRV collectively as ND issues.

1.2 Research Questions

In this thesis, we address three research questions corresponding to challenges enumerated in Section 1.1.

RQ1: How can the relationship between smart contract vulnerabilities and their detection methods be effectively established?

RQ2: How can we enhance the static detection of risks of ND state changes (caused by TOD) to Ethereum smart contract compared to existing approaches?

RQ3: How prevalent are MIV in live Ethereum smart contracts, and how can we improve the static detection of MIV and MRV in the Ethereum smart contracts compared to existing approaches?

1.3 Contributions

The contributions of this thesis are presented in a number of papers that we include in Chapter 4, and are as follows:

1. Paper I addresses *RQ1* by conducting a literature review of smart contract vulnerabilities and their detection methods. We informally identify program properties corresponding to each vulnerability. Using these

properties, we classify vulnerabilities and link them to methods that detect them statically (at pre-deployment). The survey streamlines the connection between vulnerabilities and analysis methods, providing insights to identify existing problems and solutions.

2. Papers II and III address *RQ2*. Paper II proposes an information flow-based analysis to detect RAW hazards caused by TOD. The proposed approach can detect more instances of TOD issues than existing methods but at the cost of higher false alarms. Paper III further improves the detection of RAW hazards, specifically, ND state changes caused by TOD, by tracking the state modifications from prior transactions affecting critical program parts in subsequent transactions. The evaluation reports that the proposed approach outperforms existing methods (including the approach in Paper II) regarding efficacy and execution time.

3. Papers IV and V address *RQ3*. Paper IV presents a case study identifying the prevalence of MIV in the live Ethereum smart contracts. Subsequently, it proposes an information flow-based analysis to detect MIV in Ethereum smart contracts. The proposed approach can detect more instances of MIV issues than existing methods but at the cost of higher false alarms. Paper V extends on the method proposed in Paper IV, improving the detection of MIV and further detecting MRV. The evaluation reports that the proposed approach detected five times more exploitable issues while taking 12 times less execution time than existing methods.

Table 1.1 shows ND issues (TOD, MIV, and MRV) targeted by Papers II-V.

Table 1.1: ND-issues addressed by Papers II-V.

Paper	**Nondeterministic Issues**		
	TOD	**MIV**	**MRV**
Paper II	●		
Paper III	●		
Paper IV		●	
Paper V		●	●

The rest of the thesis is organized as follows. Chapter 2 introduce ND issues in detail, Chapter 3 summarizes contributions, and Chapter 4 presents Papers I to V.

2. Background

This chapter introduces in more detail three ND issues (addressed in Papers II through V): transaction ordering dependency (TOD), missing input validation (MIV), and missing return value validation (MRV). We provide the background information on blockchains, smart contracts, and sources of nondeterminism within Ethereum in Appendix A.

2.1 Transaction Ordering Dependency (TOD)

ND-scheduling becomes problematic when multiple transactions to the same contract are batched in upcoming blocks, and at least one transaction modifies the contract's state. As a result, a contract's modified state variables may affect the execution of its operations in subsequent transactions. For example, for two transactions, a and b in a single block, if b accesses a contract's storage (global/state variables) affected by a, their execution order can result in two different contract states. This dependency of the contract final state on the order of transactions is commonly known as Transaction Ordering Dependency (TOD). ND-scheduling can also affect transactions submitted blocks apart, where some transactions can be deprived of execution, for example, in suppression attacks [11]. TOD is also akin to race conditions or frontrunning, posing security risks by allowing attackers to manipulate transaction orders.

Now, we elaborate on various forms of TOD vulnerabilities (detailed examples illustrating these vulnerabilities in Solidity contracts are given in Appendix B).

TOD on Amount (TA) of Ethers or NFTs arises when a crypto asset transfer command retrieves the transfer amount from the contract's storage, which can be modified by a previous transaction. As such, different execution orders of two transactions, one transferring crypto assets based on the value of a state variable modified by the other transaction (which introduces a RAW risk on the transfer amount), can lead to different transfer amounts. For simplicity, we refer to the state variables susceptible to RAW hazards as RAW variables.

TOD on Receiver (TR) of Ethers or NFTs is analogous to the TA, but here, the transfer recipient is retrieved from a RAW variable.

TOD on the Sender of Tokens (TST) is analogous to the TR, but here, the sender of an NFT transfer is retrieved from a RAW variable.

TOD on Transfer (TT) of Ethers or NFTs arises when a condition guarding crypto asset transfers is evaluated using RAW variables, such as sender authorization or balance checks. Consequently, these variables can introduce control dependencies, leading to nondeterministic execution of the transfers.

TOD on Receiver of `selfdestruct` (TRS) arises when the recipient of `selfdestruct` instruction is retrieved from a RAW variable.

TOD on execution of `selfdestruct` (TS) arises when a condition guarding a `selfdestruct` is evaluated using RAW variables, effectively creating a path constraint. This means that the contract's control flow—specifically, whether or not it executes the `selfdestruct` command is determined by RAW variables.

TOD on execution of `delegatecall` (TD) is analogous to the TS, but here, a condition guarding a `delegatecall` is evaluated using RAW variables.

2.2 Missing Input Validation (MIV)

Transactions submitted by users or other contracts, as well as asynchronous callbacks from Oracle services (initiated like ordinary transactions), trigger smart contract functions. Inputs with such transactions, including callback data, serve as parameters for the invoked function. Additionally, special Solidity variables like `msg.sender`, `msg.value`, and `msg.data` contain the address of the transaction sender, the transfer amount, and other inputs, respectively. Although these values are known when transactions are scheduled and remain unchanged afterward (from run to run between nodes), they can still unexpectedly influence a contract's state. For example, user input affecting the execution of `selfdestruct`, through a series of function invocations, can lead to unexpected contract termination [15]. Not validating these values before their use in such operations results in a vulnerability called Missing Input Validation (MIV). While it is common and typically a requirement to use inputs to perform certain operations, vulnerabilities often arise when such inputs are used

in critical program parts, such as transfers of crypto assets, contract termination instructions, or calls to other contracts/libraries/Oracle services, but are not validated before usage. We provide a detailed example illustrating MIV in Solidity contracts in Appendix B.

2.3 Missing Return Value Validation (MRV)

Ethers are transferable through built-in functions such as `send`, `call`, or `transfer`. NFT contracts are implemented using standards such as ERC-20 or ERC-721, which commonly implement functions like `transferFrom`, `safeTransferFrom`, or similar variants to transfer NFTs. Except for the `transfer` and safely implemented NFT transfer functions, other methods do not propagate exceptions to the caller if the transfer fails. Moreover, it is also not certain whether each external call (to a library or another) will execute successfully and return a valid result. For instance, the externally called contract may halt its execution due to factors such as running out of gas. Failure to check the return value or success status of such calls constitutes a vulnerability that permits the contract to resume its functionality under the false assumption that: i) the transfer was successful, when it may not have been, or ii) the return value is valid, despite the possibility that it is not. This vulnerability is referred to as Missing Return value Validation (MRV). A detailed example illustrating MRV in Solidity contracts is provided in Appendix B.

3. Summary of Contributions

Specific contributions of this thesis and Papers I-V are related to research questions posed in Section 1.2 and are listed as follows:

1. Paper I establishes a novel connection between smart contract vulnerabilities and their detection methods using program properties. The connection is based on the insight that if a method can detect a vulnerability by checking a particular property, it can potentially be used to detect other vulnerabilities related to the same property. Paper I classifies 52 vulnerabilities addressed in the literature using 35 properties and outlines six types of pre-deployment analysis methods that establish those properties.

2. Paper II addresses TOD vulnerabilities and proposes information flow-based analysis to track read-after-write possibilities to state variables used in Ether transfers, contract termination instructions, and conditions guarding these statements. To further enhance the detection of TOD issues, Paper III proposes an information flow-based analysis that tracks state changes in prior transactions affecting critical operations performed in subsequent transactions. As a result, Papers II and III improve the detection of risks of ND changes to smart contract states caused by TOD as compared to existing approaches.

3. Paper IV identifies the prevalence of MIV in live Ethereum smart contracts through a case study. It highlights that at least 75 live contracts on Ethereum (that received transactions at the time of inquiries) are still vulnerable to MIV, which existing methods fail to detect. Paper IV proposes an information flow-based analysis to detect MIV instances. This method is extended in Paper V, which improves the static detection of MIV and MRV by tracking the flow of unvalidated inputs and returns values into critical program parts. As a result, Papers IV and V improve the detection of risks of ND changes to smart contract states caused by inputs and return values.

School of Information Technology

ISBN: 978-91-89587-57-1 (printed)
Halmstad University Dissertations, 2024

HALMSTAD
UNIVERSITY