A Unified Specification Mining Framework for Smart Contracts

Ye Liu
li0003ye@e.ntu.edu.sg
Nanyang Technological University
Singapore

ABSTRACT

Smart contracts are self-governed computer programs that run on blockchain to facilitate asset transfer between users within a trustless environment. The absence of contract specifications hinders routine tasks, such as program understanding, debugging, testing, and verification of smart contracts. In this work, we propose a unified specification mining framework to infer specification models from past transaction histories. These include access control models describing high-level authorization rules, program invariants capturing low-level program semantics, and behavior models characterizing interaction patterns allowed by contract implementations. The extracted specification models can be used to perform conformance checking on smart contracts, with the goal of eliminating unforeseen contract quality issues.

CCS CONCEPTS

• **Software and its engineering → Software reverse engineering**.

KEYWORDS

Smart contract, specification mining.

ACM Reference Format:
Ye Liu. 2022. A Unified Specification Mining Framework for Smart Contracts. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3551349.3559512

1 INTRODUCTION

Blockchain is a distributed ledger technology, which is maintained and shared by a peer-to-peer (P2P) network. Blockchain was first introduced by Bitcoin [22] and then evolves on Ethereum [30] to support smart contract execution. Smart contracts are developed to facilitate digital asset transfer between users without a centralized authority and they are usually written in a Turing complete programming language such as Solidity [27]. Smart contracts have empowered a wide range of applications such as decentralized finance, games, NFT art markets, and so on. As of July 2022, there are more than 50 millions smart contracts deployed on Ethereum, enabling 4,073 DApps with 54.85k daily users [2, 4]. Since the notorious DAO attack [26] in 2016, smart contract security has always been the research focus. Although many approaches [11, 13, 20, 21] have been proposed to secure smart contract, it remains a big challenge to ensure the correctness with the absence of contract specification.

ASE '22, October 10–14, 2022, Rochester, MI, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3559512

Smart contracts are often developed in a rather undisciplined way. To protect assets, smart contracts tend to restrict user access by the enforcement of access control. However, Durieux et al. [9] reported that nearly 10% smart contracts may contain access control vulnerabilities. ERC20 [1] is the most popular kind of smart contracts on Ethereum but 13% ERC20 token contracts do not conform to the ERC20 standard semantics [8]. Moreover, Qin et al. [24] demonstrated how behavior models can be exploited to attack the DeFi ecosystem with flash loans. Contract specification plays a central role in describing, understanding, reasoning about smart contract behaviors, and detecting, through testing and verification, nonconformances such as correctness bugs and security violations.

There are a large body of works on specification mining. Ernst et al. [10] implemented Daikon [3] to automatically infer likely program invariants. Further, GK-tail [19] can automatically generate extended finite state machines (EFSMs) as software behavior models from runtime traces. Moreover, Iyer and Masoumzadeh [12] proposed an algorithm for mining attribute-based access control (ABAC) policies, discovering both positive and negative authorization rules simultaneously. However, there does not exist a general framework to guide the specification mining of smart contracts.

To address this problem, we propose a unified specification mining framework for smart contracts. We discuss the static artefacts and dynamic artefacts from smart contracts and its transaction history. From these artefacts, then, three different specification mining techniques: (1) *role mining*, (2) *automata learning*, and (3) *invariant detection*, are used to generate access control model, behavior model, and program invariant, respectively. Role-based access control [25] has been recommended by Openzeppelin [5] as the permission management paradigm for smart contract development. The mined access control models can be used to find permission bugs in smart contract implementations [16]. Program invariant are properties that must be preserved through the program execution. Program invariant can be used for fuzzing [28], verification [18, 23], and runtime validation [14]. Behavior model characterizes the interaction patterns allowed or expected by smart contract implementations. The behavior models can be used for conformance testing [17, 29].

2 APPROACH

2.1 Overview

Figure 1 overviews the proposed unified specification mining framework for smart contracts. The framework takes smart contracts and its transaction history as the input to generate static artefacts and dynamic artefacts. The static artefacts are minimal which include the application binary interface (ABI) specifications and contract

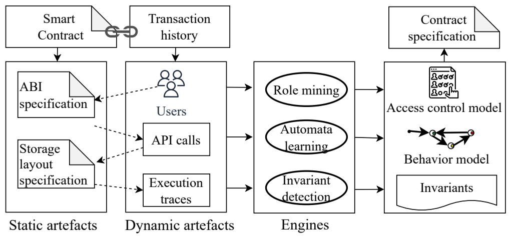

Figure 1: A unified specification mining framework.

storage layout specifications [27]. The dynamic artefacts include users, API calls and execution traces. Note that users follow the ABI specifications to call smart contract API which are public ABI functions, and the execution of API calls would modify contract state variables, thus contributing to a set of execution traces. Next, specification mining engine ensembles three techniques: (1) *role mining*, (2) *automata learning*, and (3) *invariant detection* to infer access control model, behavior model, and program invariant respectively and these will be outputted as the likely contract specifications.

2.2 Role Mining

Role mining mainly aims to address role engineering or role identification problem in applying RBAC and it discovers a set of roles from existing user access log that reflects user permission assignment. Most existing role mining techniques assume a complete user access log that contains all permissions assigned to each user. However, transaction histories of smart contracts are limited and treating it as a source of a fully-observed permission assignment will likely result in more roles than necessary and incorrect role assignments. In this section, we introduce the *partial-observation role mining problem (PORM)*, where the given permission assignment is assumed to contain only partial information.

Definition 2.1 (PORM Optimization Problem). Given a set of users U of size m, a set of permissions P of size n, and a user-permission assignment matrix UPA of size $m \times n$, PORM is to infer the unknown RBAC configuration (UA, PA), where R is a set of roles of size k, UA is a user-role assignment matrix of size $m \times k$, and PA is a permission-role assignment matrix of size $k \times n$, which satisfies,

$$\begin{aligned} \min \quad & \alpha \cdot \max_{r_i, r_j \in R} \cos\left(AFV(r_i), AFV(r_j)\right) + \beta \cdot \frac{||UA \otimes PA - UPA||_1}{||UA \otimes PA||_1}, \\ \text{s.t.} \quad & UA \otimes PA \supseteq UPA, \end{aligned} \tag{1}$$

where α and β are relative weights on the two error metrics and AFV is the role average access frequency vector proposed in [16].

2.3 Automata Learning

Contract behaviors can be modeled by a labeled state transition system [6]. A contract state transitions to another state due to the successful execution of an API call. From the historical API calls, we can infer an automaton as the behavior model of smart contract.

Definition 2.2 (Behavior Model Mining). For a smart contract, let C be the set of all valid traces of interaction with its API. Given a set of API calls and its observed finite traces $\mathcal{T} \subseteq C$, find an automaton $\mathcal{A}$ that can generate exactly/approximately the same traces in C.

Generally, $\mathcal{A}$ is an abstraction that over-approximates $\mathcal{T}$. The automaton $\mathcal{A}$ is a minimal existential abstraction [7] as follows.

Table 1: Statistics about the detected ERC20 invariants.

Invariants	bookkeeping	transfer		approve	transferFrom		
		①	②		③	④	⑤
# Contracts	126	46	25	51	2	10	6
# TPs	126	42	24	51	2	10	6
Precision	100%	91.3%	96%	100%	100%	100%	100%

Definition 2.3 (Minimal Existential Abstraction). An automaton $\mathcal{A} = (\hat{S}_0, \hat{S}, \Lambda, \hat{T})$ is the minimal existential abstraction of $\mathcal{T} = (S_0, S, \Lambda, T)$ with respect to an abstraction function $\alpha : S \rightarrow \hat{S}$ iff $\exists\ s \in S_0.\ \alpha(s) = \hat{s} \iff \hat{s} \in \hat{S}_0$ and $\exists\ (s, e, s') \in T.\ \alpha(s) = \hat{s} \wedge \alpha(s') = \hat{s}' \iff (\hat{s}, e, \hat{s}') \in \hat{T}$. where S is a set of states and $S_0 \subseteq S$ is a set of initial states; Λ is a set of events, namely, API calls, and $T \subseteq S \times \Lambda \times S$ is a relation of labeled transitions, and the definitions are similar for $\hat{S_0}$, $\hat{S}$ and $\hat{T}$.

2.4 Invariant Detection

Existing invariant detection approaches do not apply to smart contracts in that they require code instrumentation while executing a test suite dynamically to collect data traces. Due to the *gas* mechanism on blockchain, the instrumented smart contracts may behave different from the original one. In addition, few test cases exist for smart contracts, which barely cover any interesting invariant.

The proposed dynamic invariant detection technique [15] is built on Daikon [3] and we use a heuristic-based approach to recover execution traces from transaction history efficiently. The data availability on blockchain contributes to a set of execution traces caused by real world users sending contract transactions to blockchain. Each execution trace comprises the input and output to a contract transaction. Moreover, our invariant detection technique finds program invariants with a set of predefined or customized invariant templates that are unique in smart contracts.

3 RESULTS

In this section, we present the preliminary results on invariant detection. We evaluated the precision of the mined invariants on 246 real world ERC20 contracts. Table 1 shows statistics about the detected invariants, which cover seven standard ERC20 invariants: bookkeeping [28] and two `transfer` function invariants, one `approve` function invariant and three `transferFrom` function invariants [8]. The first row ("# Contracts") shows the number of contracts detected with the corresponding invariants. The second ("# TPs) and third ("Precision") rows list the number of true positives and the result precision. We successfully detected at least one ERC20 invariants in 141 unique contracts. Further, we manually construct the ground-truth whether these contracts are in line with the ERC20 standard. Overall, the results are of high accuracy.

4 CONCLUSION

In this work, we propose a unified specification mining framework to learn smart contract access control model [15], program invariant [16], and behavior model based on past transaction histories.

ACKNOWLEDGMENTS

This research is supported by the Singapore National Research Foundation under the National Satellite of Excellence in Mobile Systems Security and Cloud Security (NRF2018NCR-NSOE004-0001).

REFERENCES

[1] 2015. EIP-20: A standard interface for tokens. https://eips.ethereum.org/EIPS/eip-20.

[2] 2020. Etherscan. https://etherscan.io.

[3] 2021. Daikon. http://plse.cs.washington.edu/daikon/. The Daikon invariant detector.

[4] 2021. State of The DApps. https://www.stateofthedapps.com/zh/platforms/ethereum.

[5] 2022. OpenZeppelin. https://github.com/OpenZeppelin/openzeppelin-contracts. OpenZeppelin contracts.

[6] Sidi Mohamed Beillahi, Gabriela Ciocarlie, Michael Emmi, and Constantin Enea. 2020. Behavioral simulation for smart contracts. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 470–486.

[7] Pankaj Chauhan, Edmund Clarke, James Kukula, Samir Sapra, Helmut Veith, and Dong Wang. 2002. Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. In *International Conference on Formal Methods in Computer-Aided Design*. Springer, 33–51.

[8] Ting Chen, Yufei Zhang, Zihao Li, Xiapu Luo, Ting Wang, Rong Cao, Xiuzhuo Xiao, and Xiaosong Zhang. 2019. Tokenscope: Automatically detecting inconsistent behaviors of cryptocurrency tokens in ethereum. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 1503–1520.

[9] Thomas Durieux, João F Ferreira, Rui Abreu, and Pedro Cruz. 2020. Empirical review of automated analysis tools on 47,587 Ethereum smart contracts. In *Proceedings of the ACM/IEEE 42nd International conference on software engineering*. 530–541.

[10] Michael D Ernst, Jeff H Perkins, Philip J Guo, Stephen McCamant, Carlos Pacheco, Matthew S Tschantz, and Chen Xiao. 2007. The Daikon system for dynamic detection of likely invariants. *Science of computer programming* 69, 1-3 (2007), 35–45.

[11] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: a static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 8–15.

[12] Padmavathi Iyer and Amirreza Masoumzadeh. 2018. Mining positive and negative attribute-based access control policy rules. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*. 161–172.

[13] Bo Jiang, Ye Liu, and WK Chan. 2018. ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 259–269.

[14] Ao Li, Jemin Andrew Choi, and Fan Long. 2020. Securing smart contract with runtime validation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 438–453.

[15] Ye Liu and Yi Li. 2022. InvCon: A Dynamic Invariant Detector for Ethereum Smart Contracts. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

[16] Ye Liu, Yi Li, Shang-Wei Lin, and Cyrille Artho. 2022. Finding Permission Bugs in Smart Contracts with Role Mining. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, New York, NY, USA, 716–727.

[17] Ye Liu, Yi Li, Shang-Wei Lin, and Qiang Yan. 2020. ModCon: A Model-Based Testing Platform for Smart Contracts. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE)*.

[18] Ye Liu, Yi Li, Shang-Wei Lin, and Rong Zhao. 2020. Towards automated verification of smart contract fairness. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 666–677.

[19] Davide Lorenzoli, Leonardo Mariani, and Mauro Pezzè. 2008. Automatic generation of software behavioral models. In *Proceedings of the 30th international conference on Software engineering*. 501–510.

[20] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 254–269.

[21] Mark Mossberg, Felipe Manzano, Eric Hennenfent, Alex Groce, Gustavo Grieco, Josselin Feist, Trent Brunson, and Artem Dinaburg. 2019. Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1186–1189.

[22] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.

[23] Anton Permenev, Dimitar Dimitrov, Petar Tsankov, Dana Drachsler-Cohen, and Martin Vechev. 2020. Verx: Safety verification of smart contracts. In *2020 IEEE symposium on security and privacy (SP)*. IEEE, 1661–1677.

[24] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. 2021. Attacking the defi ecosystem with flash loans for fun and profit. In *International Conference on Financial Cryptography and Data Security*. Springer, 3–32.

[25] Ravi Sandhu, David Ferraiolo, Richard Kuhn, et al. 2000. The NIST model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, Vol. 10.

[26] David Siegel. 2016. *Understanding The DAO Attack*. https://www.coindesk.com/understanding-dao-hack-journalists

[27] Solidity 2018. Solidity. https://solidity.readthedocs.io/en/v0.5.1/.

[28] Haijun Wang, Yi Li, Shang-Wei Lin, Lei Ma, and Yang Liu. 2019. VULTRON: Catching Vulnerable Smart Contracts Once and for All. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE Press, 1–4.

[29] Yuepeng Wang, Shuvendu K Lahiri, Shuo Chen, Rong Pan, Isil Dillig, Cody Born, Immad Naseer, and Kostas Ferles. 2019. Formal verification of workflow policies for smart contracts in azure blockchain. In *Working Conference on Verified Software: Theories, Tools, and Experiments*. Springer, 87–106.

[30] Gavin Wood. 2014. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum project yellow paper* 151 (2014), 1–32.