Security Verification of Ethereum Smart Contracts with ML Taking a Free Ride from Static Analysis

Dalila Ressi*, Lorenzo Benetollo†§, Carla Piazza*, Michele Bugliesi†, Silvia Crafa ‡, and Sabina Rossi†
*University of Udine, Italy, Email: dalila.ressi@uniud.it, carla.piazza@uniud.it
†Ca' Foscari University of Venice, Italy, Email: lorenzo.benetollo@unive.it, michele.bugliesi@unive.it, sabina.rossi@unive.it
‡University of Padova, Italy, Email: silvia.crafa@unipd.it
§University of Camerino, Italy

***Abstract*—Smart contracts are compact self-executing programs running on blockchain networks and used for a range of purposes, such as enabling transactions, enforcing agreements, and managing digital assets. However, as any software system, smart contracts are vulnerable to attacks, which can result in the loss of funds or other assets.**

Developers are working to minimize the number of unsafe contracts that are deployed on blockchain networks, but the wide range of possible vulnerabilities makes assessing the security of smart contracts a challenging problem. Traditional methods like static analysis have not as yet achieved the desired accuracy and effectiveness, thus motivating a growing interest in Machine Learning approaches to the the problem.

In this presentation we discuss problems of existing vulnerability detection methods and propose possible mitigation strategies to improve their accuracy, speed, scalability, and effectiveness.

I. Introduction

Blockchain technology has gained widespread adoption in recent years as an effective infrastructure for developing decentralized applications. Among the many platforms currently available, one of the most popular is Ethereum, which first introduced smart contracts, self-executing programs that automatically enforce the rules specified within their code.

Smart contracts can be implemented for various applications: from insurance refunds to financial transactions, from corporate operations to the traceability of goods, and the protection of intellectual property [1], [2]. They are also used by Decentralized Autonomous Organizations (DAO) [3], real estate transactions [4], [5], decentralized finance [6], and in the legal industry [7].

Ethereum smart contracts are mostly written in a dedicated programming language called Solidity. They are compiled into bytecode and then deployed onto the Ethereum Virtual Machine (EVM), where they are executed in a trustless, decentralized environment.

The immutability of the blockchain ensures that once a smart contract has been published, it cannot be modified. This provides a certain level of protection against malicious entities and guarantees the fairness of the contract, but also makes it impossible to fix any bug that might be discovered in the contract's code. At the same time, being public, the blockchain is also transparent, which makes it exposed to various attacks by misbehaved miners and other adversaries. Table I summarizes some of the most common vulnerabilities of Ethereum smart contracts, arising from programming errors such as missing input validation, typecast bugs, use of untrusted inputs in security operations, unhandled exceptions, exception disorder, integer overflow / underflow [8], [9]. Detecting these problems is critical to avoid attacks that can cause millions of dollars in damages and undermine the reputation of blockchain technology (e.g., the DAO vulnerability in 2016).

TABLE I
Most recurrent vulnerabilities in smart contracts

Vulnerability	Description
Re-entrancy	A malicious contract calls back into the calling contract before the first invocation of the function is finished.
Mishandled exception	A contract is called by another contract and an exception or error is raised in the callee without being reported to the caller.
Randomness Using 'Block Hash'	Random values generated based on block hashes that can predicted by by miners.
Unprotected Ether Withdrawal	Ether withdrawn by attackers due to missing or inadequate access control.
Transaction-ordering dependence (TOD)	This vulnerability is related to the execution order of two dependant transactions that are invoking the same smart contract.
Timestamp dependency	Timestamp used as a call condition can be exploited by miners to set a particular timestamp as they can freely change it within 15 seconds.

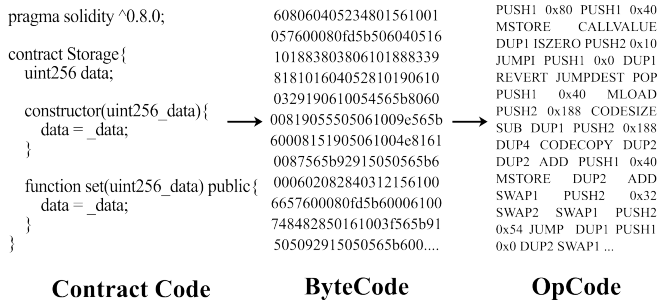
II. Detection Methods

Vulnerability detection tools are nowadays widely available from research institutions as well as corporations. They can broadly be categorized as static analyzers (sometimes complemented by a dynamic component) and Machine Learning frameworks. They operate on the smart contract source code or else directly on the contracts' bytecode / opcode retrieved from the blockchain (see Figure 1). Indeed, operating on the bytecode is not only inevitable when the source code is not publicly available, but is also the only safe choice in open platforms like the blockchain, as the bytecodes deployed on the chain may be crafted directly by an adversary to mount its attacks.

A. Formal Verification Techniques

Static analysis has long been acknowledged as an effective tool for security verification, and a number of frameworks have

Fig. 1. A simple smart contract that stores data in a private variable.



recently been developed to assess the security of Ethereum smart contracts. We list some of the most popular frameworks below. Slither [10] is a 2018 tool that employs data flow analysis to detect shadowing, uninitialized variables, re-entrancy, locked ether as wells as other arbitrary ether transfers. Mythril [11] is an analysis tool implemented in 2017 that uses symbolic execution, SMT solving, and taint analysis for Ethereum and other other blockchain platforms. Oyente [12] employs symbolic execution for analyzing a variety of properties, including transaction ordering and timestamp dependency, code re-entrancy and mihandled exceptions. Securify [13], released in 2018, also utilizes symbolic execution to detect a range of vulnerabilities by taking as input the contract bytecode and a set of security patterns.

While most existing frameworks are based on static analysis, some exploit dynamic analysis, or a mixture of both to increase precision. A notable example is SmartScan [14], an automated tool developed in 2021 that combines static and dynamic analysis specifically targeted at detecting DoS attacks.

B. Machine Learning Frameworks

Though static analysis is a powerful, and fully general approach, static analyzers are, by design, targeted at assessing specific security properties, associated with the notion of safety they are meant to enforce. As a result, while the existing tools prove effective for detecting specific vulnerabilities, their scope remains limited. The problem can be circumvented by combining different analyzers, but the practice shows that running multiple analyses on the same code may turn out to be computationally expensive and is exposed to the risk of receiving inconsistent results [15]. .

Machine Learning offers an alternative solution. Indeed, ML frameworks may leverage static analysis by employing (multiple) analyzers to create wide ranged labeled datasets which in turn may be used to train ML models for classification and vulnerability detection. Several ML experiments have been reported in the recent literature.

One method consists in transforming the source code of smart contracts (or related bytecodes) into 2D images, which are then fed to a classical Convolutional Neural Network (CNN), such as in [16] and [17]. Another popular solution is to use Graph Neural Networks (GNNs) that take as input the smart contracts with graph representation [18]–[20]. A further approach is proposed in [21], where the authors improve the privacy and security of smart contracts by restricting access privileges. [22] tested the accuracy of multiple machine learning-based approaches using XGBoost for training the models and SMOTETomek for balancing the training set, and showed they can predict six different types of vulnerabilities with an F1-score over 96%. Similarly, in [15] the authors investigate the use of various machine learning techniques to identify 16 different types of vulnerabilities. Specifically, they use SVM, Random Forest, Decision Tree, and a custom NN and compare the results and execution time to traditional techniques such as static code analyzers (Mythril and Slither [10]). The results of an experiment on about 1000 smart contracts show that machine learning frameworks can detect vulnerabilities with an average of 95% accuracy (and F1 score of 79%), within less than one second, while static techniques can require up to one hour to complete their tasks.

C. Limitations and Open Problems

As we just observed, Machine Learning seems to achieve greater performance and to guarantee protection against a wider range of vulnerabilities when compared to the currently available formal verification frameworks. On the other hand, ML suffers from an inherent scalability limitation that emerges whenever a new type of vulnerability is exposed, requiring extensive retraining for the otherwise obsolete, largely useless, ML algorithms trained on the original datasets. A more serious issue for ML frameworks is that the current static analyzers available to create the training datasets are hardly accompanied by formal soundness guarantees (with few exceptions, notably [23]), with the consequence that the currently operating ML algorithms are likely to have been (and still be) trained on mislabeled observations.

A further, independent question for current ML frameworks is their degree of precision. Establishing a standardized evaluation metrics would constitute a valuable contribution as it would make it possible to guide the choice of the most effective frameworks. Unfortunately, this is still an open, and challenging problem, as the dataset currently employed for training vary greatly both in size, in the format of the code analyzed (source code vs bytecode vs opcode), as well as in the ML technique adopted for the analysis.

D. Future Work

Our plans for future work address the limitations and open questions we just outlined, in a pragmatic attempt to mitigate them. Specifically, our initial effort will be directed towards the creation of a standardized dataset of Ethereum smart contracts, capturing a wide range of vulnerabilities, (ideally all the known ones), and including all three smart contract formats (source, byte and opcode). Running multiple analyzers on the dataset will then bring us devise a solid labeling for the dataset to be used as a solid basis for training. The standardized dataset will also provide a benchmark to held develop a standardized metrics for the evaluation of different frameworks.

REFERENCES

[1] B. Hu, Z. Zhang, J. Liu, Y. Liu, J. Yin, R. Lu, and X. Lin, "A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems," *Patterns*, vol. 2, no. 2, p. 100179, 2021.

[2] S.-Y. Lin, L. Zhang, J. Li, L.-l. Ji, and Y. Sun, "A survey of application research based on blockchain smart contract," *Wireless Networks*, vol. 28, no. 2, pp. 635–690, 2022.

[3] S. Wang, W. Ding, J. Li, Y. Yuan, L. Ouyang, and F.-Y. Wang, "Decentralized autonomous organizations: Concept, model, and applications," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 5, pp. 870–878, 2019.

[4] F. Ullah and F. Al-Turjman, "A conceptual framework for blockchain smart contract adoption to manage real estate deals in smart cities," *Neural Computing and Applications*, pp. 1–22, 2021.

[5] I. Karamitsos, M. Papadaki, N. B. Al Barghuthi *et al.*, "Design of the blockchain smart contract: A use case for real estate," *Journal of Information Security*, vol. 9, no. 03, p. 177, 2018.

[6] Y. Chen and C. Bellavitis, "Blockchain disruption and decentralized finance: The rise of decentralized business models," *Journal of Business Venturing Insights*, vol. 13, p. e00151, 2020.

[7] B. Waltl, C. Sillaber, U. Gallersdörfer, and F. Matthes, "Blockchains and smart contracts: a threat for the legal industry?" in *Business Transformation through Blockchain*. Springer, 2019, pp. 287–315.

[8] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings 6*. Springer, 2017, pp. 164–186.

[9] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H.-N. Lee, "Systematic review of security vulnerabilities in ethereum blockchain smart contract," *IEEE Access*, vol. 10, pp. 6605–6621, 2022.

[10] J. Feist, G. Grieco, and A. Groce, "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, 2019, pp. 8–15.

[11] "Mythril project," https://github.com/ConsenSys/mythril, 2019, [Online].

[12] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 254–269.

[13] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 67–82.

[14] N. F. Samreen and M. H. Alalfi, "Smartscan: An approach to detect denial of service vulnerability in ethereum smart contracts," 2021.

[15] P. Momeni, Y. Wang, and R. Samavi, "Machine learning model for smart contracts security analysis," in *2019 17th International Conference on Privacy, Security and Trust (PST)*. IEEE, 2019, pp. 1–6.

[16] T. H.-D. Huang, "Hunting the ethereum smart contract: Color-inspired inspection of potential attacks," *arXiv preprint arXiv:1807.01868*, 2018.

[17] S.-J. Hwang, S.-H. Choi, J. Shin, and Y.-H. Choi, "Codenet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection," *IEEE Access*, vol. 10, pp. 32 595–32 607, 2022.

[18] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural network." in *IJCAI*, 2020, pp. 3283–3290.

[19] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, and X. Wang, "Combining graph neural networks with expert knowledge for smart contract vulnerability detection," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[20] J. Cai, B. Li, J. Zhang, X. Sun, and B. Chen, "Combine sliced joint graph with graph neural networks for smart contract vulnerability detection," *Journal of Systems and Software*, vol. 195, p. 111550, 2023.

[21] B. D. Deebak and A.-T. Fadi, "Privacy-preserving in smart contracts using blockchain and artificial intelligence for cyber risk measurements," *Journal of Information Security and Applications*, vol. 58, p. 102749, 2021.

[22] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "Contractward: Automated vulnerability detection models for ethereum smart contracts," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1133–1144, 2020.

[23] C. Schneidewind, M. Scherer, and M. Maffei, "The good, the bad and the ugly: pitfalls and best practices in automated sound static analysis of ethereum smart contracts," in *Leveraging Applications of Formal Methods, Verification and Validation: Applications: 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20–30, 2020, Proceedings, Part III 9*. Springer, 2020, pp. 212–231.