School of Informatics

Research Methods In Security, Privacy, and Trust
Detecting Ethereum Smart Contract Security Loopholes

2187344
January 2022

Abstract

Date: Tuesday 18th January, 2022

Supervisor: Lorenzo Martinico

1 Introduction

During the predawn of the 21st century, we glimpsed a rapid technological and economic development that brought us face to face with a new technology called the blockchain. In recent years, blockchain and foremost cryptocurrencies gained much attraction due to the high monetary gains, which seem astronomical compared to traditional stock markets. It is a matter of the fact that this growth has acted as a catalyst for the technological and research eruption we have been experiencing lately. By the end of 2021, the total cryptocurrency market capitalization was 2 Trillion US Dollars[], featuring more than 16000 projects[]. Initially, the blockchain was proposed to facilitate the transfer of value completely decentralized and trust-framed over a network of peers. This was Satoshi's Nakamoto Bitcoin[] that came into existence in 2008. Since then, the research and technological trends have shifted into second-generation blockchains. Among them, the most dominant is Ethereum[]. Ethereum is a general-purpose blockchain, providing an open platform for individuals to build their applications on top of it. Their applications, i.e., Smart Contracts, are pieces of code that run decentralized on the Ethereum network. This allows developers to get involved in this new technology, creating an entirely new industry. Ethereum's ecosystem flared into a broad spectrum of Smart Contract applications, including, but not limited to, financial apps, games, digital art and music, digital voting, and patent registration.

Smart Contracts have balance measured in Ether and persistent private storage. The Smart Contract's code can manipulate changes in the program's variables and storage. On Ethereum, Smart Contracts code is in EVM[] (Ethereum Virtual Machine), a Turing complete stack-based bytecode language spanning 144 OP codes[]. Nonetheless, developers can define their code in a high-level language such as Solidity[], compiled to EVM bytecode afterward. A new transaction invocation can cause the execution of the contract's code by receiving inputs and producing outputs. Typically, when a user wants to trigger a Smart Contract needs to create a new transaction and pay some fees[]. Transaction fees depend on the code being executed on the Ethereum network.

Beyond any doubt, the Ethereum blockchain is used to manage digital assets reflecting a considerable value. Hence Smart Contracts gained the interest of malign entities. Numerous attackers perform attacks on the Ethereum applications, with the ultimate goal of stealing Ether from them. One of the most severe attacks was the infamous DAO[]. In more detail, in 2016, an autonomous decentralized organization was founded to direct a venture capital fund. This organization was formed on the Ethereum blockchain, and over 11000 investors deposited into a Smart Contract over 150 Million US Dollars[] considering the Ether price back then. Then, an unprecedented attack[] was performed on that Smart Contract, resulting in 50 Million US Dollars loss. Another critical incident happened with the Parity Wallet[], where the attack [] led to freezing 150 Million US Dollars, in terms of Ether, impermanently.

Nevertheless, it is typical for any piece of code to have bugs. Likewise, we expect Smart Contracts to behave in the same way. Potentially, five main reasons make Smart Contracts vulnerable. We might blame developers for not wholly understanding the blockchain development stack[smart check]. Also, Solidity is a relatively new language with many limitations and challenges. Thus, it is hard for developers to use it[smart check]. We consider blockchain immutability as another non-helping characteristic. Any application deployed on the Ethereum network can not be modified, so software fixes are not immediately doable[contract fuzzer]. A public blockchain allows financially motivated attackers to use their online pseudonymity to exploit any software bug[smart check]. Finally, we cannot control the Smart Contract execution environment since the network runs in a decentralized fashion[smart check]. For this reason,

Smart Contract developers need to detect possible vulnerabilities in their code before going live. A Smart Contract Auditing tool would benefit both users and developers. A developer might use the tool to detect any code issues before deploying the Smart Contract. At the same time, users can utilize the tool to check if the Smart Contract they are depositing Ether is safe and does behave maliciously.

Indeed, such tools came into existence in 2016, with the first one being OYENTE[]. Since then, academic researchers have invested time and effort in expanding this research field, developing a plethora o tools. Some of them are general, trying to detect any vulnerability, while some others focus on a few of them. Also, they employ different analysis techniques, such as Static Analysis, Dynamic Analysis, Symbolic Execution, Fuzzing, and Machine Learning. Furthermore, researchers attempted to document and classify many of these vulnerabilities according to their behavior or functionality.

The contribution of this work is three-fold. Initially, we document and classify Smart Contract vulnerabilities reported across the most prominent studies. Next, we present the research trends of developing a Smart Contract vulnerability framework throughout the years, giving an overview of the techniques used and directions taken. Finally, we elaborate on the result reliability and scalability of the most paramount frameworks.

The structure of the following literature review has as follows ...

2 Methodology

In this literature review, we explore the most prominent research attempts towards creating an effective vulnerability detection tool for Ethereum Smart Contracts. The method used to filter relevant studies comprised both bottom-up and top-down strategies. Their combination significantly accelerated identifying adequate quality papers to include in this literature review.

During the bottom-up stage, we sought several papers related to frameworks that identify Smart Contract vulnerabilities. To achieve that, we employed trustworthy academic search engines such as *Google Scholar* and *IEEE Explorer*. We noticed that most of the retrieved writings had a joint primary related work, which guided us to discover this research topic's essence paper[1].

Subsequently, we aimed to find any derivative studies related to the paper arisen in the previous phase. To do so, we used a graph representation tool[2] that links relevant papers. This mechanism allowed us to identify remarkable research articles rapidly. Afterward, we manually inspected the search results and included the most reliable in our under investigation list.

Namely, we collected forty-seven papers, but we only consider nineteen of them in this literature review. The selection criteria span the context of the studies and their research contribution. We measure their contribution according to their citations and release year. According to their publication year, studies are assigned a weight ranging from 1 to 4. Papers in the span of 2016-2017, 2018, 2019, 2020 receive 1, 2, 4, 8 points, respectively, for each citation they hold. Using this metric, we evaluate the studies shown in Table 2.

Study	Year	Cit.	Score	Ref.
Making Smart Contracts Smarter	2016	1445	1445	[1]
A Survey of Attacks on Ethereum Smart Contracts (SoK)	2017	1176	1176	[3]
Securify: Practical Security Analysis of Smart Contracts	2018	436	872	[4]
VerX: Safety Verification of Smart Contracts	2020	103	824	[5]
ZEUS: Analyzing Safety of Smart Contracts	2018	399	798	[6]
Finding The Greedy, Prodigal, and Suicidal Contracts at Scale	2018	357	714	[7]
SmartCheck: Static Analysis of Ethereum Smart Contracts	2018	291	582	[8]
Formal Verification of Smart Contracts	2016	525	525	[9]
ContractFuzzer:Fuzzing Smart Contracts for Vulnerability Detection	2018	233	466	[10]
Slither: A Static Analysis Framework For Smart	2019	109	436	[11]
MadMax:Surviving Out-of-Gas Conditions in Ethereum Smart Contracts	2018	213	426	[12]
Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts	2019	91	364	[13]
teether:Gnawing at Ethereum to Automatically Exploit Smart Contracts	2018	173	345	[14]
Vandal:A Scalable Security Analysis Framework for Smart Contracts	2018	154	308	[15]
ReGuard: Finding Reentrancy Bugs in Smart Contracts	2018	122	244	[7]
Ethereum Smart Contracts: Vulnerabilities and their Classifications	2020	12	96	[16]
ETHPLOIT:From Fuzzing to Efficient Exploit Generation against Smart Contracts	2020	10	80	[17]
GasFuzzer: Fuzzing Ethereum Smart Contract Binaries to Expose Gas-Oriented Exception Security Vulnerabilities	2020	8	64	[18]
SoliAudit: Smart Contract Vulnerability Assessment Based on Machine Learning and Fuzz Testing	2019	13	52	[19]

Table 1: List of papers that are analyzed in this literature review, sorted by their score

3 Literature Review

3.1 Smart Contract Attack Surface

3.2 Static Analysis

3.3 Dynamic Analysis

3.4 Symbolic Execution

3.5 Fuzzing

3.6 The Machine Learning Approach

4 Summary & Conclusion

References

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

[2] Find and explore academic papers. `https://www.connectedpapers.com/`. Accessed: 2021-11-04.

[3] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts (sok). In *International conference on principles of security and trust*, pages 164–186. Springer, 2017.

[4] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 67–82, 2018.

[5] Anton Permenev, Dimitar Dimitrov, Petar Tsankov, Dana Drachsler-Cohen, and Martin Vechev. Verx: Safety verification of smart contracts. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1661–1677. IEEE, 2020.

[6] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. Zeus: Analyzing safety of smart contracts. In *Ndss*, pages 1–12, 2018.

[7] Chao Liu, Han Liu, Zhao Cao, Zhong Chen, Bangdao Chen, and Bill Roscoe. Reguard: finding reentrancy bugs in smart contracts. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pages 65–68. IEEE, 2018.

[8] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. Smartcheck: Static analysis of ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, pages 9–16, 2018.

[9] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, et al. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM workshop on programming languages and analysis for security*, pages 91–96, 2016.

[10] Bo Jiang, Ye Liu, and WK Chan. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 259–269. IEEE, 2018.

[11] Josselin Feist, Gustavo Grieco, and Alex 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)*, pages 8–15. IEEE, 2019.

[12] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. Madmax: Surviving out-of-gas conditions in ethereum smart contracts. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):1–27, 2018.

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

[14] Johannes Krupp and Christian Rossow. teether: Gnawing at ethereum to automatically exploit smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1317–1333, 2018.

[15] Lexi Brent, Anton Jurisevic, Michael Kong, Eric Liu, Francois Gauthier, Vincent Gramoli, Ralph Holz, and Bernhard Scholz. Vandal: A scalable security analysis framework for smart contracts. *arXiv preprint arXiv:1809.03981*, 2018.

[16] Zulfiqar Ali Khan and Akbar Siami Namin. Ethereum smart contracts: Vulnerabilities and their classifications. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 1–10. IEEE, 2020.

[17] Qingzhao Zhang, Yizhuo Wang, Juanru Li, and Siqi Ma. Ethploit: From fuzzing to efficient exploit generation against smart contracts. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 116–126. IEEE, 2020.

[18] Imran Ashraf, Xiaoxue Ma, Bo Jiang, and Wing Kwong Chan. Gasfuzzer: Fuzzing ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities. *IEEE Access*, 8:99552–99564, 2020.

[19] Jian-Wei Liao, Tsung-Ta Tsai, Chia-Kang He, and Chin-Wei Tien. Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 458–465. IEEE, 2019.

[20] Ivica Nikolić, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 653–663, 2018.