Understanding Security Audits on Blockchain

Zhiju Yang

Seattle University, zhijuyang@seattleu.edu

Gaoyuan Man

Arizona State University, gman2@asu.edu

Songqing Yue

University of Wisconsin - Platteville, yues@uwplatt.edu

Hacks on blockchains are prevalent nowadays because a wide range of vulnerabilities exists in smart contracts. To deal with the vulnerabilities, security auditing has been emerging and widely adopted for manual code inspection. However, little is known about how manual security auditing impacts the blockchain community. In this work, we investigated security audits on blockchain by answering three research questions covering the type, severity, and resolve status of issues in the audit reports. We in total collected 2,421 security audit reports consisting of the details of 28,782 security issues. We found that manual security auditing has superior merit over static analysis tools in identifying logic-related issues, considering 27.65% of all findings are logical issues. We also found that 87.28% of 2,421 projects have at least one centralization issue that breaks the decentralization primitive of blockchain. Moreover, while 21.17% of issues are of major or critical severity, we surprisingly found that 61.83% of all issues were unresolved or partially resolved. We concluded that while manual security auditing can help identify security vulnerabilities uniquely, project developers need to take serious actions to address and secure their smart contracts.

CCS CONCEPTS • Security and privacy; Software and application security; Software security engineering

Additional Keywords and Phrases: Blockchain, Smart contract, Security, Audit, Measurement

1 INTRODUCTION

During the past decade, blockchain has been revolutionizing rapidly and drawing substantial attention across the world. Essentially, blockchain is a decentralized digital ledger that stores immutable and permissionless information. Since the pioneer introduction of Bitcoin in 2008 [1], a variety of blockchains, such as Ethereum [2], Solana [3], and Aptos [4], have been designed and developed. These blockchains innovatively transform a wave of industries such as health care, supply chain, manufacturing, etc. Among all the blockchain applications, cryptocurrency has gained tangible success in recent years. According to CoinMarketCap [5], the cryptocurrency market capitalization is as high as 975 billion dollars as of August 2022. Specifically, decentralized exchange (DEX), non-fungible tokens (NFTs), decentralized finance (DeFi), and stablecoins are among the most popular cryptocurrency applications, which have more than hundreds of millions of users [6].

Pain does exist while blockchain prospers and goes viral. Except for the Bitcoin blockchain, most blockchains support the store and execution of a set of agreements written in the computer program, which is known as a smart contract[1]. It is a fundamental building block that allows developers to create various decentralized apps and tokens. Unfortunately, due to the nature of smart contract development, a wide variety

[1]In this paper, we use the term *smart contract* in a broad context that generally represents the user-defined program on a blockchain. Note that different blockchains might have their specific terms such as *module* on Aptos and *program* on Solana.

of vulnerabilities, e.g., reentrancy [7] and arithmetic overflow and underflow [8], could exist in a smart contract and result in substantial financial loss for its users [9]. Hackers have already stolen nearly $2 billion worth of cryptocurrency in 2022 according to the latest report [10]. For example, a security breach on the Ronin Network is exploited by hackers and over $624 million worth of cryptocurrency was stolen in March 2022 [11].

To deal with the vulnerabilities of smart contracts, researchers and developers have proposed various defense solutions and tools. For example, static analysis tools, such as Slither [12], Securify [13], SmartCheck [14], Mythril [15], and SmartBugs [16], can automate the vulnerability detection of smart contracts; security-orientated programming languages, such as Move [17] and Obsidian [18], are introduced to develop more secure smart contracts; scam detection solutions investigated in [19] can detect anomaly transactions (e.g., phishing, rug pulls, and fraud) of smart contracts.

Along with these efforts in securing smart contracts, manual security auditing is gradually being adopted by many projects as an indispensable procedure to detect any security vulnerability before deploying the smart contract. An overview of the manual security auditing process is summarized in Figure 1. In contrast to static analysis, the most superior advantage of manual security auditing lies in the phase of manual code inspection, in which security auditors understand and examine the business logic of each individual smart contract, and identify security holes based on their intelligence and experience [24]. Many companies and organizations currently provide smart contract auditing services [25]. For example, CertiK, one of the leading blockchain security companies, has audited over 2,000 projects since it was founded in 2018 [26].



Figure 1: An overview of the security audit process.

Despite the ever-growing of blockchain security companies and their auditing services, little is known about how manual security auditing impacts the blockchain community. Thus, we set out to uncover security audits on blockchain by investigating the following three research questions:

- RQ1: What typical issues are identified in manual security auditing?
- RQ2: How severe are the identified issues?
- RQ3: Are the identified issues resolved?

To answer these questions, we collected 2,421 security audit reports from CertiK that consist of the details of 28,782 security issues. We found that the top three security issues are the logical issue, centralization issue, and volatile code issue, which account for 27.65%, 17.84%, and 14.97%, respectively. We confirmed that manual security auditing has superior merit over static analysis tools in identifying logic-related issues. We also found that 87.28% of 2,421 projects have at least one centralization issue that breaks the decentralization primitive of blockchain. From the perspective of the issue severity, we found that nearly half (i.e., 48.09%) of issues are informational and 21.17% of issues are of major or critical severity. We quantified issue severity using the accumulated CVSS score and found that the average score is 23.57, indicating the seriousness of security vulnerabilities in those projects. Lastly and surprisingly, we found that 61.83% of 28,782 reported security issues were unresolved or partially resolved, and 855 out of 2,421 projects did not resolve any issues in the audit report. We concluded that while manual security auditing can help identify security vulnerabilities uniquely, project developers need to take serious actions to address and secure their smart contracts.

2 RELATED WORK

Researchers have systematically surveyed and summarized the security and privacy challenges on different blockchains and smart contracts [27, 28, 29, 30]. Towards securing blockchain applications and smart contracts, tremendous efforts have been put into different blockchain facets.

In the space of smart contract analysis, several automated vulnerability detection tools have been proposed. Tsankov et al. [13] proposed Securify which is a scalable and fully automated security analyzer of Ethereum smart contracts. It symbolically analyzes the dependency graph of the given smart contract and identifies violation patterns for security hole detection. Tikhomirov et al. [14] proposed SmartCheck - an extensible static analysis tool that translates smart contract source code into an XML-based representation for pattern recognition. To facilitate and ease the automated analysis, testing, and debugging using static analysis tools, Ferreira et al. [16] proposed an easy-to-use execution framework called SmartBugs in 2020. Furthermore, Feist et al. [12] proposed Slither which runs a suite of more than 70 vulnerability detectors. Slither is still actively developed and is widely used as of August 2022. Alongside these automated tools, a wide range of static analysis approaches have also been investigated, for example, fuzz testing [31], [32], symbolic execution [33, 34, 35], model checking [36], and formal verification [37, 38, 39].

Besides, security-orientated programming languages have been implemented or leveraged to develop more secure smart contracts. For example, Move [17], a new platform-agnostic language, is developed by Facebook engineers for writing secure, verified, yet flexible smart contracts; Vyper [40] consolidates smart contract development with many security features such as overflow checking, decidability, and strong typing. In addition, best practices for smart contract development have also been investigated and recommended in [41, 42, 43].

3 DATA COLLECTION AND DATASET

Nearly one hundred companies and organizations currently provide smart contract auditing services according to this compiled provider list [25]. Among all these auditing providers, CertiK is one of the leading blockchain security companies and has audited over 2,000 projects since it was founded in 2018 [26]. To the best of our knowledge, CertiK completed auditing the greatest number of projects and is far ahead of its competitors[2]. Therefore, because of its dominance in the blockchain security auditing area, we choose CertiK as the data source to answer our research questions as described in Section 1.

As shown in Figure 1, a final audit report is usually created at the end of the auditing phase. The report contains the description of any identified vulnerabilities, corresponding severity, suggested remediations, and auditee's action. Most reports are publicly available for the purpose of project marketing and user confidence build. Moreover, CertiK shares and maintains all the project details and audit reports on its Security Leaderboard webpage. Therefore, we set out to collect all these shared project details and audit reports in August 2022.

In total, we collected 2,798 audit reports ranging from June 2018 to August 2022. After filtering out penetration testing reports and duplicate audit reports, a dataset of 2,421 valid audit reports is created for our work. Most audit reports include the information of the blockchain or platform where the audited project resides. Table 1 summarizes the top 10 blockchains/platforms which the audited projects build upon. It is

[2]Solidproof audited the second most projects (i.e., 597) as of August 2022 [44].

clear that most audited projects come from two dominant blockchains, i.e., Ethereum and Binance Smart Chain, which have 1,066 and 1,035 audited projects, respectively. While EVM-based blockchains[3] dominate this top-ten list, a non-negligible portion of projects lives on emerging blockchains such as Solana, Tron, and Cosmos. Note that some projects interact with multiple blockchains for cross-chain bridge services and hence one project could contribute to the count multiple times. For example, the iSwap protocol interacts with 14 blockchains for cross-chain asset exchange.

Table 1: The top 10 blockchains/platforms where the audited projects build upon

Blockchain/Platform	Count	Blockchain/Platform	Count	Blockchain/Platform	Count
Ethereum	1,066	Huobi Eco Chain	44	Fantom	14
BNB Smart Chain	1,035	Solana	20	Tron	12
Polygon	79	Avalanche	18	OKEx	10
				Cosmos	6

In the collected 2,421 audit reports, there are in total 28,782 security issues (also referred to as findings) identified by auditors. Statistically, about 12 security issues were identified per audited project on average and the median value is 10. It is worth mentioning that a maximum of 181 issues were identified from 79 files of the Delta Finance project[4].

4 ANALYSIS AND RESULTS

4.1 What typical issues are identified in audit reports?

Each audit report contains detailed issues along with their type, severity, and resolve status. After extracting all the issue types from the report, we identified in total 12 different types of issues disclosed in 2,421 audit reports. Figure 2 summarizes them along with the number of issues of each type.

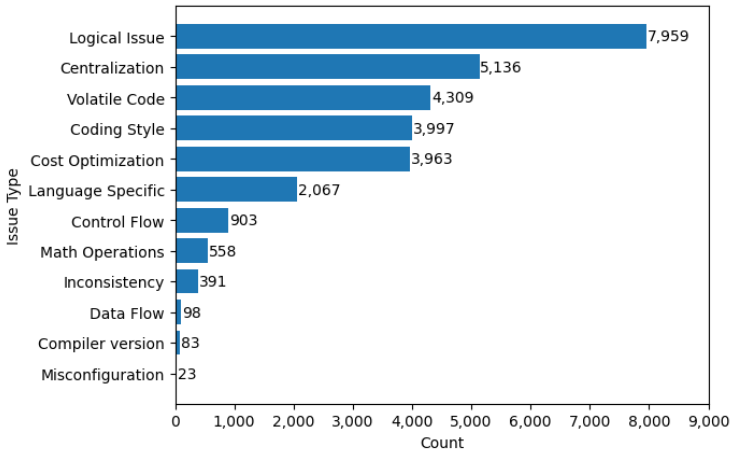


Figure 2: Twelve types of issues (sorted by count) disclosed in the 2,421 audit reports

[3]EVM stands for Ethereum virtual machine, which is the core software that executes smart contracts and computes the state of the Ethereum network.

[4]The corresponding report can be accessed at https://certik-public-assets.s3.amazonaws.com/CertiK-Audit-for-Delta-Finance--v5.pdf

The most common issue is the logical issue, followed by the centralization issue and the volatile code issue. Based on the definition in each report, a logical issue refers to a fault in program logic that could cause the smart contract to operate incorrectly; a centralization issue refers to a program component that acts against the nature of decentralization (e.g., a privileged access role that is able to relocate all users' tokens); and, a volatile code issue refers to problematic code that behaves unexpectedly on certain edge cases and may result in a vulnerability. Therefore, it is safe to conclude that manual security auditing has superior merit over static analysis tools in identifying logic-related issues, considering the fact that the latter such as Slither [12] cannot understand the smart contract in a human-like manner.

As shown in Figure 2, 7,959 out of 28,782 (i.e., 27.65%) findings are logical issues. From the perspective of the individual project, 1,816 out of 2,421 (i.e., 75.01%) projects have at least one logical issue identified, which indicates the importance and necessity of manual security auditing. To better understand the logical issue, we now take the finding DTC-11 in the audit report for Dopple Finance[5] as an example. The "DopToken" is created in this project and is supposed to have a mint limit of `manualMintLimit`. As shown in the code below, the variable `manualMinted` is used to record the amount of already minted "DopToken" and is checked against `manualMintLimit` in every mint operation to make sure the limit is not exceeded. However, the value of `manualMinted` is never updated, hence the `require` check will always be passed and unlimited "DopToken" can be minted without any restriction. This is a critical logical issue because it could significantly break the designed tokenomics (as one can perceive from the involved variables) by allowing unlimited token supply. To remediate this issue, the Dopple Finance team revised the `manualMint` function following the auditor's suggestion, which adds `_amount` to `manualMinted` in every mint operation to keep its value and logic correctness. Note that without a human-like understanding of the implementation, it is unlikely for static analysis tools to identify such an issue.

```
function manualMint(address _to, uint256 _amount) public onlyOwner {
    require(manualMinted <= manualMintLimit, "mint limit exceeded");
    mint(_to, _amount);
}
```

The centralization issue is the second most prevalent problem, i.e., 5,136 out of 28,782 (17.81%) in all the audited projects. Furthermore, from the perspective of the individual project, 2,113 out of 2,421 (i.e., 87.28%) projects have at least one centralization issue identified. This is a particularly astonishing finding because one of the attractive features of blockchain is decentralization, but most of these projects cannot offer it.

To have a deeper understanding of the centralization issue, we sampled 100 such issues from 100 projects and summarized them into three categories as shown in Table 2. We found that 68% of the sampled centralization issues are relevant to the existence of the privileged role in the smart contract. In most cases, those smart contracts allow a privileged owner account to modify the sensitive states. For example, as identified in the audit finding INP-01 of project InfinityDAO[6], a privileged owner role has the permission to modify multiple sensitive states of the smart contract, of which the most concerning privilege is that the owner can transfer any assets in the smart contract to any account. Furthermore, we found that 26% of the sampled projects have a centralization issue of arbitrary token distribution. That is, during the initial token distribution process, all the tokens are sent to an externally owned account (EOA) and such a delegated EOA account is

[5]The corresponding report can be accessed at https://www.certik.com/projects/dopplefinance
[6]The corresponding report can be accessed at https://www.certik.com/projects/infinitydao

responsible for distributing tokens. This raises a serious concern that the initial token distribution is fully charged by a particular party without obtaining the consensus of the community. In addition, we found that the centralized fee account exists in 6% of the sampled projects. Such an account is used to collect various fees such as the swapping fee and liquidity reward. However, such an account is under the control of a particular party, which can withdraw the assets in the account at any time without any consensus.

Table 2: Three categories of 100 sampled centralization issues in 100 projects

Category	Percentage	Description
Privileged Role	68%	A privileged account that can modify the sensitive states of a smart contract
Arbitrary Token Distribution	26%	A special account that can distribute tokens arbitrarily
Centralized Fee Account	6%	A fee collection account that can be withdrawn from by the particular owner

4.2 How severe are the identified issues?

To answer this research question, we leveraged the severity information of each issue in the audit report. Overall, there are five different severities, which are critical, major, medium, minor, and information in the order of seriousness. This glossary is adapted from the standard common vulnerabilities and exposures (CVE) [45], which uses the common vulnerability scoring system (CVSS) to assess a vulnerability and assign a severity on a scale of one to ten. Figure 3 presents the percentage of identified issues in each severity category. As we can see, nearly half (i.e., 48.09%) of issues are informational, indicating that the corresponding vulnerabilities have relatively little severity. For example, the issue of unlocked compiler version has the informational severity; a smart contract with this issue can be compiled using a different version of Solidity compiler, which could hence result in bytecode ambiguity and compiler-specific vulnerabilities since certain old compilers have known vulnerabilities [46]. Furthermore, we found that 22.97% of issues are minor, and 21.17% of issues are major or critical. The logical issue discussed in Section IV-A is an example of a critical issue, and most centralization issues are considered as having major severity.

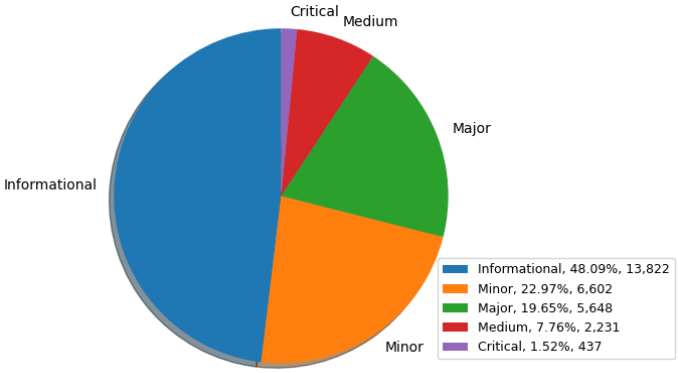


Figure 3: The percentage of identified issues in each severity category.

Table 3 details the severity and resolve status summary of each issue type. A pair of numbers in each severity column represents the amount of resolved issues and total issues. We found that the dominant issue types such as logical issue and centralization tend to have greater severity in general, and issues of cost optimization and language specific are usually informational. This, again, indicates the importance and superior merit of manual security auditing, considering the seriousness of the identified issues.

Table 3: The detailed severity and resolve status summary of each issue type

Issue Type	Total	Severity				
		Critical Resolved/Total	Major Resolved/Total	Medium Resolved/Total	Minor Resolved/Total	Informational Resolved/Total
Logical Issue	7,959	215/258	581/1,027	604/1,268	1,091/2,778	970/2,597
Centralization	5,136	29/113	373/4,370	48/320	39/294	10/38
Volatile Code	4,309	23/33	74/163	155/358	868/2,420	469/1,331
Coding Style	3,997	0/0	1/6	14/28	77/159	1,639/3,803
Cost Optimization	3,963	0/0	0/5	11/22	74/180	1,556/3,754
Language Specific	2,067	5/10	13/31	28/56	59/128	840/1,840
Control Flow	903	16/24	40/83	47/121	110/375	74/296
Math Operations	558	2/3	18/29	52/86	121/262	82/178
Inconsistency	391	2/2	15/20	30/40	57/112	115/216
Data Flow	98	1/1	4/13	5/12	14/26	20/46
Compiler Version	83	11/11	10/11	1/4	16/27	13/30
Misconfiguration	23	1/1	0/1	2/3	10/16	2/2

To quantify the severity of each project, we calculated their CVSS score based on the issue severity in the report. Since such a score is not included in the audit report, we conservatively calculate it by assigning a lower-bound base score [47] to each severity and summing all scores up to get an accumulated CVSS score. Note that to take informational (which is not included in the CVE standard) issues into consideration, we assign 0.1 to an informational issue and 0.5 to a minor issue. We strictly keep the lower-bound base score for medium, major, and critical issues, which is 4.0, 7.0, and 9.0, respectively. Figure 4 presents the quantitative severity view of all projects, Ethereum projects, and Solana projects. Note that a greater CVSS score indicates the project is more vulnerable and outliers are removed for better boxplot presentation. We found that for all the 2,421 projects, the average accumulated CVSS score is 23.57 as shown by the green line in the left boxplot; 50% of these projects have a CVSS score between 8.20 and 29.90. When comparing projects across blockchains, as shown in the right two boxplots, we found that Solana projects have an average score of 17.99 while Ethereum projects have an average score of 24.64; it is more likely for Ethereum projects to have more vulnerabilities as indicated by the wide range whisker in the middle boxplot. One of the potential reasons is that most Solana projects were written in Rust which has a focus on memory safety.

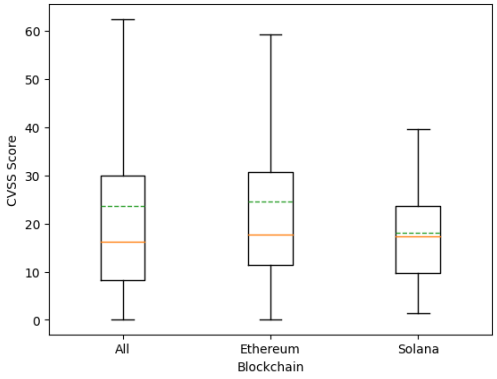


Figure 4: The quantitative severity view of all projects, Ethereum projects, and Solana projects.

Overall, these results indicate that blockchain projects have various vulnerabilities of different severities and the manual security auditing can help identify them and potentially consolidate the project.

4.3 Are the identified issues resolved?

While one might fairly expect that most of the identified security issues would be addressed seriously, we surprisingly found that only 31.17% of all the identified 28,782 issues are resolved, leaving the rest 61.83% of issues unresolved or partially resolved. Table 3 provides a summary of the resolve status of each issue type. We found that the vast majority (i.e., 90.28% of 5,136) centralization issues and over half (i.e., 56.51% of 7,959) logical issues were not resolved. Generally, the more severe the issue, the higher the chance it gets resolved. However, it is still concerning that 132 of 437 critical-level issues were not addressed. Moreover, we found that 855 out of 2,421 projects did not resolve any issues in the audit report.

With these concerning results, we investigated the reasons why so many issues were not addressed. By examining the auditee's responses to issues in the audit report, we observed and summarized the following common ones: (1) acknowledge the issue but address it in the future, (2) acknowledge the issue but cannot resolve it since the smart contract is already deployed, (3) decline the issue, and (4) no response received.

In more details, we found that the first one is very typical in response to a centralization issue. There are usually three types of remediations to this issue, which are (1) a short-term solution that uses timelock and multisignature to delay and control sensitive operations, (2) a long-term solution that uses timelock and decentralized autonomous organization (DAO) to delay and decentralize sensitive operations, and (3) a permanent solution that removes any privileged roles and functions. Since any of these solutions requires a considerable engineering effort and some projects are intended to be centralized, many issues are just acknowledged but not resolved in the current codebase. Next, we surprisingly found that the second one is also common in audited projects. For example, the DefiHoldings[7] team responded that they cannot resolve identified issues since their smart contracts have already been deployed on the blockchain. This reflects an abnormal yet popular trend in the current blockchain community, that is, many projects leverage the security audit report as an endorsement and a marketing scheme, instead of taking any actions to treat the audit details seriously. Besides, we found that 85 issues are false positives and hence declined by the auditee.

5 CONCLUSION

In this work, we investigated security audits on blockchain by answering three research questions covering the type, severity, and resolve status of issues in audit reports. We collected 2,421 security audit reports consisting of the details of 28,782 security issues. We found that the top three security issues are the logical issue, centralization issue, and volatile code issue, which account for 27.65%, 17.84%, and 14.97%, respectively. We confirmed that manual security auditing has superior merit over static analysis tools in identifying logic-related issues. We also found that 87.28% of 2,421 projects have at least one centralization issue that breaks the decentralization primitive of blockchain. Besides, we quantified the issue severity using the accumulated CVSS score and found that the average score is 23.57, indicating the seriousness of vulnerabilities in those projects. Moreover, while 21.17% of issues are major or critical, we surprisingly found that 61.83% of issues were unresolved or partially resolved and 855 out of 2,421 projects did not resolve any issues. We conclude that while manual security auditing can help identify security vulnerabilities uniquely, project developers need to take serious actions to address and secure their smart contracts.

[7]The corresponding report can be accessed at https://www.certik.com/projects/defi-holdings

REFERENCES

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Decentralized Business Review, p. 21260, 2008.

[2] V. Buterin et al., “A next-generation smart contract and decentralized application platform,” white paper, vol. 3, no. 37, 2014.

[3] “Solana Blockchain,” 2022. https://solana.com/.

[4] “Aptos Blockchain,” 2022. https://aptoslabs.com/.

[5] “CoinMarketCap,” 2022. https://coinmarketcap.com/.

[6] “Defi Llama,” 2022. https://defillama.com/.

[7] M. Rodler, W. Li, G. O. Karame, and L. Davi, “Sereum: Protecting existing smart contracts against re-entrancy attacks,” arXiv preprint arXiv:1812.05934, 2018.

[8] “National Vulnerability Database,” 2022. https://nvd.nist.gov/vuln/detail/CVE-2018-10299

[9] A. Mense and M. Flatscher, “Security vulnerabilities in ethereum smart contracts,” in Proceedings of the 20th international conference on information integration and web-based applications & services, pp. 375–380, 2018.

[10] “Chainalysis Mid-year Crypto Crime Update,” 2022. https://blog.chainalysis.com/reports/crypto-crime-midyear-update-2022/.

[11] “Ronin Network Hack,” 2022. https://rekt.news/ronin-rekt/.

[12] 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), pp. 8–15, IEEE, 2019.

[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, pp. 67–82, 2018.

[14] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, “Smartcheck: Static analysis of ethereum smart contracts,” in Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, pp. 9–16, 2018.

[15] “Mythril,” 2022. https://github.com/ConsenSys/mythril.

[16] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, “Smartbugs: A framework to analyze solidity smart contracts,” in Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, pp. 1349–1352, 2020.

[17] S. Blackshear, E. Cheng, D. L. Dill, V. Gao, B. Maurer, T. Nowacki, A. Pott, S. Qadeer, D. R. Rain, S. Sezer, et al., “Move: A language with programmable resources,” Libra Assoc, 2019.

[18] M. Coblenz, “Obsidian: a safer blockchain programming language,” in 2017 IEEE/ACM 39th international conference on software engineering companion (ICSE-C), pp. 97–99, IEEE, 2017.

[19] B. Mazorra, V. Adan, and V. Daza, “Do not rug on me: Leveraging machine learning techniques for automated scam detection,” Mathematics, vol. 10, no. 6, p. 949, 2022.

[20] P. N. Sureshbhai, P. Bhattacharya, and S. Tanwar, “Karuna: A blockchain-based sentiment analysis framework for fraud cryptocurrency schemes,” in 2020 IEEE International Conference on Communications Workshops (ICC Workshops), pp. 1–6, IEEE, 2020.

[21] J. Wu, Q. Yuan, D. Lin, W. You, W. Chen, C. Chen, and Z. Zheng, “Who are the phishers? phishing scam detection on ethereum via network embedding,” IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2020.

[22] “Token Sniffer,” 2022. https://tokensniffer.com/.

[23] “Rug Pull Detector,” 2022. http://rugpulldetector.com/.

[24] D. He, Z. Deng, Y. Zhang, S. Chan, Y. Cheng, and N. Guizani, “Smart contract vulnerability analysis and security audit,” IEEE Network, vol. 34, no. 5, pp. 276–282, 2020.

[25] “Discover the World’s Top Smart Contract Auditing Companies,” 2022. https://smartcontractaudits.com/audit-providers/companies/1.

[26] “CertiK Security Audit,” 2022. https://www.certik.com/products/security-audit.

[27] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on ethereum smart contracts (sok),” in International conference on principles of security and trust, pp. 164–186, Springer, 2017.

[28] R. Zhang, R. Xue, and L. Liu, “Security and privacy on blockchain,” ACM Computing Surveys (CSUR), vol. 52, no. 3, pp. 1–34, 2019.

[29] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, “A survey on the security of blockchain systems,” Future Generation Computer Systems, vol. 107, pp. 841–853, 2020.

[30] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, “Smart contract development: Challenges and opportunities,” IEEE Transactions on Software Engineering, vol. 47, no. 10, pp. 2084–2106, 2019.

[31] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, “Reguard: finding reentrancy bugs in smart contracts,” in 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), pp. 65–68, IEEE, 2018.

[32] B. Jiang, Y. Liu, and W. K. Chan, “Contractfuzzer: Fuzzing smart contracts for vulnerability detection,” in 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 259–269, IEEE, 2018.

[33] J. Chang, B. Gao, H. Xiao, J. Sun, Y. Cai, and Z. Yang, “scompile: Critical path identification and analysis for smart contracts,” in International Conference on Formal Engineering Methods, pp. 286–304, Springer, 2019.

[34] J. Krupp and C. Rossow, “{teEther}: Gnawing at ethereum to automatically exploit smart contracts,” in 27th USENIX Security

Symposium (USENIX Security 18), pp. 1317–1333, 2018.

[35] 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, pp. 254–269, 2016.

[36] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, “Zeus: analyzing safety of smart contracts.,” in Ndss, pp. 1–12, 2018.

[37] E. Hildenbrandt, M. Saxena, N. Rodrigues, X. Zhu, P. Daian, D. Guth, B. Moore, D. Park, Y. Zhang, A. Stefanescu, et al., “Kevm: A complete formal semantics of the ethereum virtual machine,” in 2018 IEEE 31st Computer Security Foundations Symposium (CSF), pp. 204–217, IEEE, 2018.

[38] D. Park, Y. Zhang, M. Saxena, P. Daian, and G. Ros ̧u, “A formal verification tool for ethereum vm bytecode,” in Proceedings of the 2018 26th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp. 912–915, 2018.

[39] I. Grishchenko, M. Maffei, and C. Schneidewind, “Foundations and tools for the static analysis of ethereum smart contracts,” in International Conference on Computer Aided Verification, pp. 51–78, Springer, 2018.

[40] Vyper Documentation,” 2022. https://vyper.readthedocs.io/en/stable/.

[41] “Ethereum Smart Contract Security Best Practices,” 2022. https://consensys.github.io/smart-contract-best-practices/.

[42] “Thinking About Smart Contract Security,” 2022. https://blog.ethereum.org/2016/06/19/thinking-smart-contract-security.

[43] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, “Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab,” in International conference on financial cryptography and data security, pp. 79–94, Springer, 2016.

[44] “SolidProof,” 2022. https://app.solidproof.io/projects.

[45] “Common Vulnerabilities and Exposures (CVE),” 2022. https://en.wikipedia.org/wiki/Common Vulnerabilities and Exposures.

[46] H. Chen, M. Pendleton, L. Njilla, and S. Xu, “A survey on ethereum systems security: Vulnerabilities, attacks, and defenses,” ACM Computing Surveys (CSUR), vol. 53, no. 3, pp. 1–43, 2020.

[47] “NVD Vulnerability Severity Ratings,” 2022. https://nvd.nist.gov/vuln-metrics/cvss