Analysis Of The Relationship Between Smart Contracts' Categories and Vulnerabilities

Giacomo Ibba
Dep. of Mathematics and Computer Science
University Of Cagliari
Cagliari, Italy
giacomo.ibba@unica.it

Marco Ortu
Dep. of Business and Economics Sciences
University of Cagliari
Cagliari, Italy
marco.ortu@unica.it

2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) | 978-1-6654-3786-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SANER53432.2022.00143

***Abstract*—Smart Contracts are general-purpose programs that provide a higher level of security than traditional contracts and reduce other transaction costs associated with the bargaining practice, as they are executed in a Blockchain infrastructure. Developers use smart contracts to build their tokens and set up gambling games, crowd sales, ICO, and many others domains of application. The security of Smart Contracts is also crucial, as SCs at the very core level, move money. In recent years, researchers have provided a set of known vulnerabilities that afflict SCs. This study analyzed the relationship between the SC domain of application, domain category, and known vulnerabilities. We categorized the SC using the topic modeling on a curated dataset of SC annotated with know vulnerabilities. Indeed, we found that a certain category of SC is strongly associated with specific vulnerabilities.**

***Keywords*—Blockchain, Smart Contract, Ethereum, Vulnerabilities, Topic Modeling, Smart Contracts Categories**

I. Introduction

Since Ethereum's smart contracts spread in the computer science field, developers started to exploit their properties and peculiarities to build several general-purpose programs. Popular trends of smart contract development include money investment, certification, and gambling game programs. Another recent trend includes Non-Fungible-Token programs [26], whose aim is to certify a digital asset as unique and therefore not interchangeable. Apart from the ones listed previously, there are other categories of smart contracts. Previous work [2] proposed a smart contracts taxonomy of 5 types:

- Financial: contracts managing, gathering, or distributing money.
- Notary: contracts that certify ownership and provenance of data.
- Game: contracts that implement gambling games and skill games.
- Wallet: contracts that, in general, simplify the interaction with the blockchain.
- Library: contracts that implement general-purpose operations.

The proposed taxonomy dates back to 2017, which overlaps with the wide diffusion of smart contracts.

The work also shows the relationship between category and design patterns [28], [14]. Still, since the taxonomy is old, developers could spot new categories and design patterns of Ethereum smart contracts. It is unlikely that developers and user participants will spot contracts implementing only a library in 2021 since, nowadays, these contracts are part of more sophisticated programs. Moreover, essential design patterns like the non-fungible-token started to spread at the end of 2018 and the beginning of 2019, so a new analysis of smart contracts categories is required.

Another crucial part of smart contracts programming is to avoid writing vulnerable programs [20]. Many programs are designed specifically to perform money transfer operations or act like banks, keeping Ether safe. A bug or security leak in such programs could lead to an irreversible loss of Ether. The most famous attack is the "DAO attack," [16] which is the perfect example of how a security leak in a smart contract could lead to money loss. For this very reason, it is crucial to prevent any Ether loss due to security leaks by spotting them and correcting the vulnerable pattern.

Several vulnerabilities may affect a smart contract, but some are worse than others regarding potential Ether loss. Denial of Service (DoS) and reentrancy are two of the most dangerous: reentrancy exploits the features of Ethereum contracts and utilizes the code of other external programs; an attacker can hijack these external calls whereby they force the program to execute malicious code. DoS consists of leaving a smart contract inoperable for a short period or, in some cases, permanently. There are several ways to lead a program inoperable, like looping through externally manipulated mapping or arrays or allowing only the smart contract's owner to execute some operations. Other vulnerabilities include:

- Arithmetic overflows and underflows.
- Unexpected code execution due to the DELEGATECALL opcode.
- Unexpected loss of Ether due to improper use of the suicide/selfdestruct function.
- Block timestamps manipulation.
- Improper iterations over array or mapping with a big amount of elements.

Several works already explore smart contracts vulnerabilities, but no work studies the correlation between categories and security leaks. For example, given the class of Financial contracts, which are the principal vulnerabilities affecting

978-1-6654-3786-8/22/$31.00 ©2022 IEEE
DOI 10.1109/SANER53432.2022.00143

these programs? Apart from this, in literature, there is not a massive analysis of specific smart contracts categories, and since previous works are relatively old, and during this time, it could be possible to detect new design patterns inside the Ethereum blockchain. So, this work aims to categorize a considerable amount of Ethereum smart contracts and then analyze the correlation between categories and vulnerabilities.

The work enriches the existing literature in several ways. From the statistics point of view, it helps to understand which are the most used categories of smart contracts, giving a better understanding of the blockchain technology, particularly the Ethereum one. Also, it helps to understand the correlation between types and vulnerabilities and on which classes of programs developers should focus their attention to avoid the most dangerous security leaks.

II. State Of Art

Smart contracts classification is a task of great interest since the Ethereum technology spread. Indeed, researchers carried out several works. One possible approach consists in performing a transaction-based analysis to spot behaviour patterns [10]. Other methods capture grammar rules and context information from source code [23], while others use the bytecode to classify contracts [21], [15], [24]. Other works are more specific to classify contracts vulnerabilities and bugs. For example, Reguard [13] parses a smart contract code to an intermediate representation and then performs a source-to-source transformation from the intermediate model to C++. As output, the tool returns a bug report with all the possible patterns of reentrancy. Other tools such as Osiris [1] find integer bugs in Ethereum smart contracts. At the same time, ContractFuzzer [12] defines test oracles to detect security vulnerabilities and exploits the log of the EVM to report such vulnerabilities. Other tools help detect Ethereum contracts vulnerabilities [5], [6], [27], [9], but most of them are built to check a specific exposure or only a subset of them. Given the number of work that developers carried out, it is proof that Ethereum contracts classification and vulnerabilities detection are tasks of high interest for the research community. Still, no works in literature explore the correlation between a given smart contract category and the possible vulnerabilities affecting the program, which is the aim of this research.

III. Research Methodology

The following chapter presents our research methodology. We collected many contracts covering different years (from 2017 to 2021). At the end of the process of programs collection, we were able to retrieve more than 105k contracts by joining three different datasets of interest. To be more precise, the initial size of our dataset was over 140K samples, but after merging the contracts with the same address but different source code, we reduced the dataset size to 105K.

A. Dataset

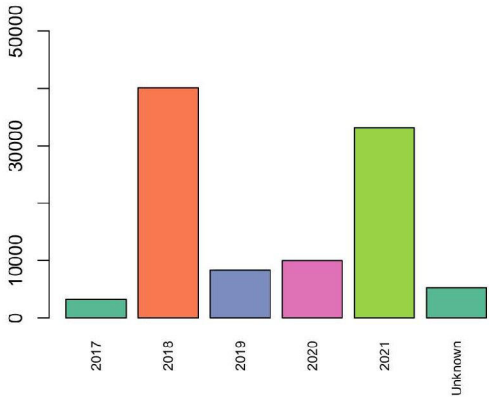

Fig. 1: Number of Smart Contracts Sample Per Year

Our dataset is a collection of contracts retrieved from other existing datasets. The first dataset from which we collected our contracts is SmartCorpus [19], an organized, reasoned, and up-to-date repository where developers and researchers can efficiently and systematically access Solidity source code and other metadata about Ethereum smart contracts. We took advantage also of the SmartSanctuary [17] dataset, which is available on Reddit and contains only Ethereum verified smart contracts. The interesting central aspect of this dataset is that it is constantly updated and includes programs for both test nets (Goerli, Kovan, and Ropstent). The last and most relevant dataset we exploited is Smart Bugs [8], which contains vulnerable smart contracts and reports the vulnerability to which the contract is exposed for a subset of samples. This dataset is particularly crucial for our analysis since it contains contracts with reported vulnerabilities; this feature allows us to study the correlation between categories and exposures.

By joining three datasets encapsulating many smart contracts, we could find duplicates. Therefore, in the preprocessing steps, we removed any copy from our dataset, where we consider a counterpart as a contract having the same source code and address. After removing the duplicates, the dataset's size decreased to 100040 samples.

Figure 1 shows the number of smart contracts samples of our dataset per year after the duplicates removal process. Our range goes from 2017 to 2021, and the most represented years are 2018 and 2021. Having a lot of programs deployed in a time interval between 2018 and 2021 increased the representativeness of our dataset since 2018 was a crucial year for new Solidity programs spreading and original design patterns developing. Nevertheless, we lack a lot of 2019 and 2015, and 2016 contracts, so we could enrich our sample and its representativity by adding the programs missed in our dataset.

B. Topic Modeling

To spot contracts categories, we took advantage of source codes and comments made by developers. The main idea is that the terms used to set variable names, function names, struct names, and comments could help to spot the contract's purpose. Another interesting observation is that some programs have a well-defined structure in terms of source code; indeed, many developers recycle the source code of already deployed contracts to build their programs. For example, observing tokens, gambling games, investments, and others are pretty much the same looking at the code and structure, and this feature could help in our analysis.

These considerations led us to choose natural language processing techniques, particularly topic modeling techniques [25] and the Latent Dirichlet Allocation (LDA) model [4], to categorize our samples. Before talking about the LDA model, it should be emphasized that we are dealing with code and not human language. To deal with this particular feature of our problem, we appended to stopwords all the Solidity and assembly keywords (since it could be possible to write assembly code inside a Solidity smart contract). Apart from language keywords, developers can write code using snake case or camel case as writing style; to deal with snake case, we split all the words separated by underscores. For camel case variables, we divided attached words, in which the first word begins with a lowercase letter, during the following ones with a capital letter.

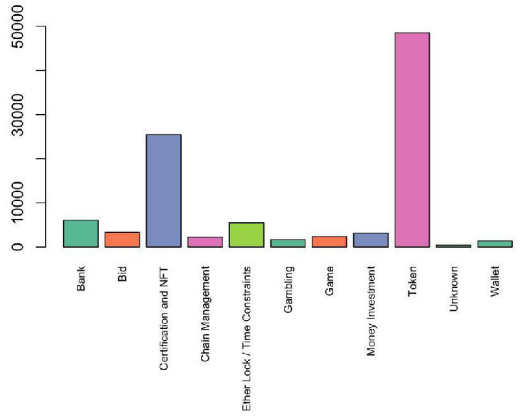

Fig. 2: Spotted categories after the seeded LDA analysis

It is fundamental to find the optimal number of topics before using the LDA model to find the smart contracts categories. To find the optimal number of types, we computed the c_v measure [22], which we used to calculate the topic coherence, which measures the score of a single topic by measuring the degree of semantic similarity between high-scoring words in the topic. We found the optimal model by computing the coherence values augmenting the number of topics. If the coherence score seems to keep increasing, it is a reasonable choice to pick the model that gave the highest CV before the curve flattens out or before a significant drop. After the coherence score calculation, we found that the optimal topic number is between 10 and 15.

Eventually, we took advantage of a seeded LDA to help the model in topic modeling. The idea is to associate some terms to a specific topic. Suppose the LDA model categorizes a document that contains the keywords related to the guide topic.

In that case, it will assign the highest topic probability for that document with the associated guide topic. For example, if a document contains terms like 'play,' 'player,' 'bet,' 'dice,' 'pot,' 'prize,' etc., it is almost sure that the smart contract is implementing a gambling game.

After analyzing our sample with the seeded LDA, the model joined four topics with others; in particular, bids smart contracts aggregate ICO [7] and Crowdsales, gambling includes oracle smart contracts, and finally, investment contracts include Ponzi schemes [11]. Figure 2 shows the categories of smart contracts and the number of samples per category inside our dataset.

- **Bank** contracts implement a virtual bank, allowing participants to store their Ether. A user can withdraw his money at any moment.
- **Bid** contracts implement auctions to buy a specific good. These contracts also include ICO and Crowdsales.
- **Certification and NFT** (CNFT) are contracts certifying the authenticity and the ownership of a digital asset.
- **Chain Management** (CM) programs implement chain operations.
- **Ether Lock / Time Constraints** (ELTC) are like Bank contracts, but users cannot retrieve their Ether at any time. They must comply with time constraints.
- **Gambling** are programs implementing a gambling game.
- **Game** category differs from Gambling, because includes skill games, role playing games, etc.
- The **Money Investment** category includes smart contracts where a user can invest his money to earn. This category also includes Ponzi schemes, which are scams.
- **Token** contracts implement tokens and specific operations to deal with them.
- **Wallet** programs act like wallets, performing transactions, storing Ether, etc.
- We considered the documents unrelated to any of the categories above as **Unknown** (for example, Hello World, nonsense operations, and useless mathematical operations programs).

The following section explores the correlation between category and vulnerability.

IV. Results

We considered the curated Smart Bugs dataset that includes 143 already annotated Smart Contracts with known vulnerabilities according to DASP taxonomy [1]. The considered

[1]https://dasp.co/

Category	Top 5 Keywords
Token	amount supply spender spend burn
Certification and NFT	ownership state proposal sale beneficiary
Bid	round buy sell need debt
Bank	amount fee lock math withdraw
Etherlock / Time Constraints	amount time get start end
Money Investment	investor lock distribute team invest
Wallet	transaction target freeze refund wallet
Gambling	bet prize hash oraclizequery limit
Game	player role price key game
Chain Management	list storage remove date operator
Unknown	?

TABLE I: Categories and top 5 keywords per topic

SC categories are: ELTC, Gambling, Game, Token, Wallet, Bank, CM, and CNFT. The Money Investment and Unknown categories were excluded from the analysis because no sample with an already known vulnerability was available in our dataset. On the other side, the considered vulnerabilities are: Time Manipulation (TM), Arithmetic, Bad Randomness (BR), Unchecked Low Level Calls (ULLC), Access Control (AC), Denial Of Service (DoS) and Reentrancy. We already discussed the Reentrancy and DoS vulnerabilities; the others consist in:

- TM vulnerabilities are due to the possibility for miners to adjust timestamps slightly, which can be pretty dangerous if block timestamps are misused in smart contracts.
- Arithmetic vulnerabilities include arithmetic overflows and underflows, which occur when the program executes an operation that requires a fixed size variable to store a number (or piece of data) that is outside the range of the variable's data type.
- The BR vulnerability is quite complex to understand deeply [2]. It is not possible to implement randomness directly since the blockchain nodes must converge at the same result. Therefore, it is evident that randomness must be achieved through an external source, but it could be possible to use hashes, blocknumber, gas limit, and timestamps to achieve randomness. The problem is that these variables are controlled by the miner who mines the block, so this is not truly a random implementation and could lead to several issues.
- ULLC consists of harmful use of the call() function of the address type, with which you can call several potential dangerous functions.
- AC vulnerabilities are due to bad design of which user can perform potentially dangerous operations.

Figure 3 shows the contingency table of the two observed characteristics of the Smart Contracts: Category and Vulnerability. This figure shows a graphical matrix where each cell contains a dot whose size reflects the relative magnitude of the corresponding joint frequency. We can see that the observed joint frequencies (Category, Vulnerability) tend to be distributed on specific (Category, Vulnerability) cells, indicating a stochastic independence.

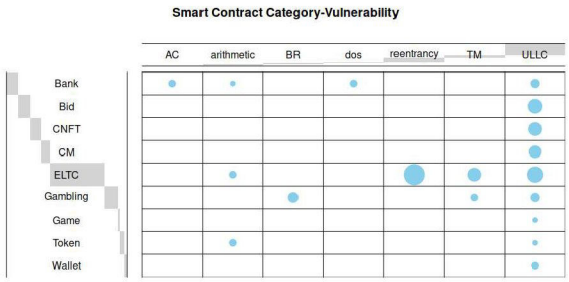

Fig. 3: Graphical Representation Of The Category VS Vulnerability Contingency Table.

This evidence can be further remarked using the Mosaic Plot shown in Figure 4. Here, blue values indicate that the observed joint frequencies are *higher* than the expected frequencies and red values indicate that the observed joint frequencies are *lower* than the expected frequencies. In this case, we can see that all the associations are positive.

To test the independence hypothesis, we used a Chi-square test in order to examine whether rows and columns of the

[2]https://medium.com/hackernoon/hackpedia-16-solidity-hacks-vulnerabilities-their-fixes-and-real-world-examples-f3210eba5148

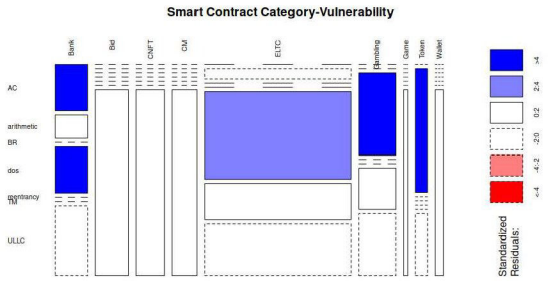

Fig. 4: Mosaic Plot of The Category VS Vulnerability Contingency Table.

contingency table are statistically significantly associated. For this test, the Null and Alternative hypothesis are:

H_0: Row and column variables of the contingency table are independent.

H_1: Row and column variables are dependent.

For each cell of the contingency table, we have to calculate the expected value under Null hypothesis. For a given cell, the expected value of the joint frequency is calculated as follows:

$$e = \frac{row\ sum * col\ sum}{\#\ observations}$$

The Chi-Square statistic is calculated as:

$$\chi^2 = \sum \frac{(o-e)^2}{e}$$

We obtained a $\chi^2 = 131.54$ and a $p-value = 1.048e-09$, we thus reject the Null hypothesis, and we conclude that the two variables, Category and Vulnerability, are not independent. This implies that their association is statistically significant.

The contribution of each cell to the total Chi-square score can be calculated using the Chi-square residual statistic for each cell: $r = \frac{o-e}{\sqrt{e}}$, the so-called Pearson residuals. Figure 5 shows the Pearson residuals, where circle size is proportional to the cell contribution. Here it is very important the sign of the standardized residuals, which is critical to interpret the relationship between rows (Category) and columns (Vulnerability). The sign of the standardized residual is interpreted as follows.

- Positive residuals are highlighted in blue. Positive values in cells indicate a positive association (attraction) between the relevant row and column variables.
- Negative residuals are shown in red. This suggests that the associated row and column variables have a repulsion (negative relationship).

Figure 5 shows that the all major associations are positive.

The contribution of each cell is obtained by the following ratio: $contr = \frac{r^2}{\chi^2}$, where r is the cell Pearson residual.

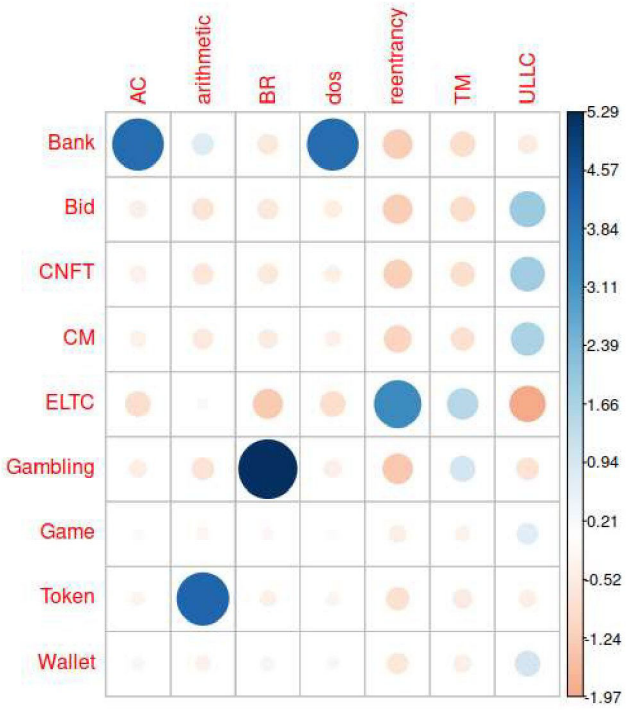

Fig. 5: Correction Plot Of The χ^2 Pearsons Residuals.

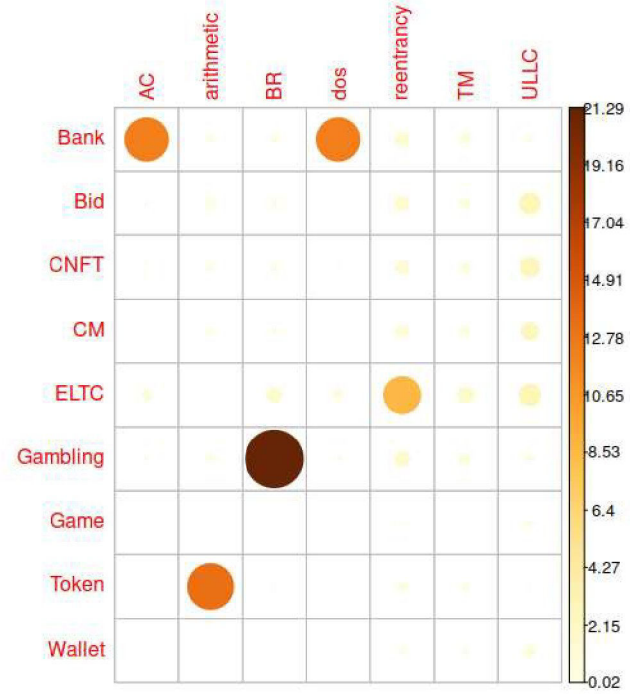

Fig. 6: Contribution of each (Category-Vulnerability) cell to the total χ^2 Statistic.

Finally, Figure 6 highlights the contribution of each cell, giving some indication of the nature of the dependency between rows (Category) and columns (Vulnerability) of the contingency table. Our results show that Smart Contracts that belongs to the Gambling Smart Contracts are *strongly* associated to Bad Randomness (they contribute to 21.29% of the total χ^2), while, Token and Arithmetic (13.32% of the total χ^2) and Ether Lock/Time Constraint, Bank Category and Access Control vulnerability (12.32% of the total χ^2), Bank and DoS (12.31% of the total χ^2) and Reentrancy (8.69% of the total χ^2) are *frequently* associated to each others. These findings can be useful in the development phase, considering the specific category of the Smart Contract and focusing resources in specific categories of vulnerabilities.

V. Conclusion and Future Works

The security of Smart Contracts is a critical aspect for the success of such technologies. In this study, we analysed the relationship of the domain application of Smart Contracts and vulnerabilities. We categorized Smart Contracts in specific domains of application using NLP techniques, we found that these Smart Contract categories are statistically significant associated with known vulnerabilities.

In particular, we found that Gambling Smart Contracts are *strongly* associated to Bad Randomness (they contribute to 21.29% of the total χ^2), while, Token and Arithmetic (13.32% of the total χ^2) and Ether Lock/Time Constraint, Bank Category and Access Control vulnerability (12.32% of the total χ^2), Bank and DoS (12.31% of the total χ^2) and Reentrancy (8.69% of the total χ^2) are *frequently* associated to each others.

In the future, we plan to use statistical analysis tools, such as [8] to analyse the vulnerabilities of a larger corpus of Smart Contracts ($\sim 130K$) to confirm our findings. Another research direction is to understand the influence of development communities on the category and vulnerabilities of Smart Contracts [18], [3].

To increase the representativeness of our dataset, we aim to collect more contracts. Notably, we want to increase the number of samples for 2019 and 2017, for which we have a few programs. Also, we don't have any smart contracts for 2015 and 2016. Despite being the early years of the Ethereum blockchain and before the smart contract trend spread, developers could have carried out exciting contracts in this time interval, so it might be worth retrieving programs in this range. Moreover, our dataset of 100040 contracts contains only 143 already annotated programs for which the vulnerability is known. In future work, we aim to label as many as possible of the other contracts of our dataset by taking advantage of the already existing vulnerabilities spotting tools.

References

[1] *Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts*, 12 2018.

[2] Massimo Bartoletti and Livio Pompianu. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *International conference on financial cryptography and data security*, pages 494–509. Springer, 2017.

[3] Silvia Bartolucci, Giuseppe Destefanis, Marco Ortu, Nicola Uras, Michele Marchesi, and Roberto Tonelli. The butterfly "affect": impact of development practices on cryptocurrency prices. *EPJ Data Science*, 9(1):21, 2020.

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[5] Monika Di Angelo and Gernot Salzer. A survey of tools for analyzing ethereum smart contracts. In *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*. IEEE, 2019.

[6] 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, 2019.

[7] Gianni Fenu, Lodovica Marchesi, Michele Marchesi, and Roberto Tonelli. The ico phenomenon and its relationships with ethereum smart contract environment. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 26–32, 2018.

[8] João F Ferreira, Pedro Cruz, Thomas Durieux, and Rui Abreu. Smartbugs: a framework to analyze solidity smart contracts. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1349–1352, 2020.

[9] Roberta Galici, Laura Ordile, Michele Marchesi, Andrea Pinna, and Roberto Tonelli. Applying the etl process to blockchain data. prospect and findings. *Information*, 11(4):204, 2020.

[10] Teng Hu, Xiaolei Liu, Ting Chen, Xiaosong Zhang, Xiaoming Huang, Weina Niu, Jiazhong Lu, Kun Zhou, and Yuan Liu. Transaction-based classification and detection approach for ethereum smart contract. *Information Processing and Management*, 58(2):102462, 2021.

[11] Giacomo Ibba, Giuseppe Antonio Pierro, and Marco Di Francesco. Evaluating machine-learning techniques for detecting smart ponzi schemes. In *2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 34–40, 2021.

[12] 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.

[13] 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, 2018.

[14] Yue Liu, Qinghua Lu, Xiwei Xu, Liming Zhu, and Haonan Yao. Applying design patterns in smart contracts. In Shiping Chen, Harry Wang, and Liang-Jie Zhang, editors, *Blockchain – ICBC 2018*, pages 92–106, Cham, 2018. Springer International Publishing.

[15] Michele Marchesi, Andrea Pinna, Francesco Pisu, and Roberto Tonelli. Crypto-trading. rechargeable token-based smart energy market enabled by blockchain and iot technology. In *European Conference on Parallel Processing*, pages 166–178. Springer, 2019.

[16] Muhammad Izhar Mehar, C. Shier, Alan Giambattista, Elgar Gong, G. Fletcher, Ryan Sanayhie, Henry M. Kim, and M. Laskowski. Understanding a revolutionary and flawed grand experiment in blockchain: The dao attack. *J. Cases Inf. Technol.*, 21:19–32, 2019.

[17] Martin Ortner and Shayan Eskandari. Smart contract sanctuary.

[18] Marco Ortu, Tracy Hall, Michele Marchesi, Roberto Tonelli, David Bowes, and Giuseppe Destefanis. Mining communication patterns in software development: A github analysis. In *Proceedings of the 14th international conference on predictive models and data analytics in software engineering*, pages 70–79, 2018.

[19] Giuseppe Antonio Pierro, Roberto Tonelli, and Michele Marchesi. Smart-corpus: an organized repository of ethereum smart contracts source code and metrics, 2020.

[20] Purathani Praitheeshan, Lei Pan, Jiangshan Yu, Joseph Liu, and Robin Doss. Security analysis methods on ethereum smart contract vulnerabilities: A survey, 2020.

[21] Chaochen Shi, Yong Xiang, Robin Ram Mohan Doss, Jiangshan Yu, Keshav Sood, and Longxiang Gao. A bytecode-based approach for smart contract classification. *arXiv preprint arXiv:2106.15497*, 2021.

[22] Shaheen Syed and Marco Spruit. Full-text or abstract? examining topic coherence scores using latent dirichlet allocation. In *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 165–174, 2017.

[23] Gang Tian, Qibo Wang, Yi Zhao, Lantian Guo, Zhonglin Sun, and Liangyu Lv. Smart contract classification with a bi-lstm based approach. *IEEE Access*, 8:43806–43816, 2020.

[24] Roberto Tonelli, Andrea Pinna, Gavina Baralla, and Simona Ibba. Ethereum smart contracts as blockchain-oriented microservices. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, pages 1–2, 2018.

[25] Hanna M. Wallach. Topic modeling: Beyond bag-of-words. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 977–984, New York, NY, USA, 2006. Association for Computing Machinery.

[26] Qin Wang, Rujia Li, Qi Wang, and Shiping Chen. Non-fungible token (nft): Overview, evaluation, opportunities and challenges. *arXiv preprint arXiv:2105.07447*, 2021.

[27] Wei Wang, Jingjing Song, Guangquan Xu, Yidong Li, Hao Wang, and Chunhua Su. Contractward: Automated vulnerability detection models for ethereum smart contracts. *IEEE Transactions on Network Science and Engineering*, 2020.

[28] Maximilian Wöhrer and Uwe Zdun. Design patterns for smart contracts in the ethereum ecosystem. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1513–1520, 2018.