scientific reports

Check for updates

OPEN

Taxonomic insights into ethereum smart contracts by linking application categories to security vulnerabilities

Marco Ortu[1]✉, Giacomo Ibba[2], Giuseppe Destefanis[3], Claudio Conversano[1] & Roberto Tonelli[2]

The expansion of smart contracts on the Ethereum blockchain has created a diverse ecosystem of decentralized applications. This growth, however, poses challenges in classifying and securing these contracts. Existing research often separately addresses either classification or vulnerability detection, without a comprehensive analysis of how contract types are related to security risks. Our study addresses this gap by developing a taxonomy of smart contracts and examining the potential vulnerabilities associated with each category. We use the Latent Dirichlet Allocation (LDA) model to analyze a dataset of over 100,040 Ethereum smart contracts, which is notably larger than those used in previous studies. Our analysis categorizes these contracts into eleven groups, with five primary categories: Notary, Token, Game, Financial, and Blockchain interaction. This categorization sheds light on the various functions and applications of smart contracts in today's blockchain environment. In response to the growing need for better security in smart contract development, we also investigate the link between these categories and common vulnerabilities. Our results identify specific vulnerabilities associated with different contract types, providing valuable insights for developers and auditors. This relationship between contract categories and vulnerabilities is a new contribution to the field, as it has not been thoroughly explored in previous research. Our findings offer a detailed taxonomy of smart contracts and practical recommendations for enhancing security. By understanding how contract categories correlate with vulnerabilities, developers can implement more effective security measures, and auditors can better prioritize their reviews. This study advances both academic knowledge of smart contracts and practical strategies for securing decentralized applications on the Ethereum platform.

Ethereum[1], launched in 2015, built upon the foundation laid by Bitcoin[2]. While Bitcoin had already introduced basic smart contract functionality, supporting applications like prediction markets and rudimentary exchanges, Ethereum significantly expanded the possibilities of blockchain technology. Its key innovation, the Ethereum Virtual Machine (EVM), provided a more accessible framework for developing complex, Turing-complete smart contracts. This advancement, coupled with substantial backing from its creators, attracted a growing community of developers who saw potential in Ethereum's more flexible approach to blockchain programming. As a result, Ethereum quickly became a central platform for decentralized application development[3], complementing rather than replacing Bitcoin's role in the cryptocurrency ecosystem[4–7].

Ethereum's smart contracts, which execute automatically based on conditions, have made decentralized application creation more accessible. This highlights the importance of developing strong methods for categorising and analysing the increasing number of contracts due to their growing complexity and potential in various fields.

The evolution of SC usage has resulted in a substantial number of contracts on the Ethereum blockchain, rendering manual inspection unfeasible[8]. High-profile security breaches, such as the DAO incident, where a significant amount of Ether was stolen due to a contract loophole, spotlight the need for enhanced security measures[9]. The DAO[10], or Decentralized Autonomous Organization, was an ambitious project built on the Ethereum blockchain, aiming to create a venture capital fund that operated without traditional management structures. In 2016, a vulnerability in its smart contract code was exploited, allowing an attacker to siphon

[1]Department of Business and Economics Sciences, University of Cagliari, Viale Fra Ignazio 17, Cagliari, Italy. [2]Department of Computer Science and Mathematics, University of Cagliari, Via Porcell 4, Cagliari, Italy. [3]Department of Computer Science, Brunel University London, London, UK. ✉email: marco.ortu@unica.it

approximately $50 million worth of Ether from the fund. This incident not only highlighted the risks associated with smart contract vulnerabilities[11–14] but also led to a controversial hard fork of the Ethereum blockchain to recover the stolen funds.

Moreover, the overwhelming number of smart contracts complicates their efficient management and classification, making it harder to spot contracts that are either at risk or engaging in deceitful activities. These challenges underline the urgent need for advanced analytical methods to understand and organize smart contracts, a gap our research aims to fill.

While we focus on smart contract categorization and vulnerability analysis[15,16], recent advancements, such as blockchain-based authentication for mobile IoT devices, also contribute to the broader effort of enhancing blockchain ecosystem security (e.g., Wen et al.[17]).

The rapid proliferation of smart contracts on the Ethereum blockchain has led also to a diverse ecosystem of decentralized applications[18–21]. However, this growth has also introduced challenges in understanding, categorizing, and securing these contracts. The complexity and variety of smart contracts make it difficult for developers, auditors, and users to comprehend the landscape of applications and their associated risks. This study is motivated by the need to:

(I) Develop a taxonomy of smart contracts that reflects the current state of the Ethereum ecosystem. (II) Understand the evolution of smart contract applications over time. (II) Identify potential vulnerabilities associated with different categories of smart contracts.

To address these needs, we pose the following research questions:

- RQ1: What are the primary categories of smart contracts currently deployed on the Ethereum blockchain?
- RQ2: How have smart contract applications evolved since the inception of Ethereum?
- RQ3: Is there a correlation between specific smart contract categories and their vulnerability to certain types of security risks?

This study makes the following key contributions to the field of smart contract analysis:

- A data-driven taxonomy of Ethereum smart contracts: We present a comprehensive categorization of smart contracts based on the analysis of over 100,000 unique contracts, providing insights into the current landscape of decentralized applications.
- Temporal analysis of smart contract evolution: We track the development and popularity trends of different smart contract categories over time, offering a historical perspective on the Ethereum ecosystem's growth.
- Linking contract categories to vulnerabilities: We establish correlations between specific smart contract categories and their susceptibility to various types of vulnerabilities, providing valuable insights for developers and auditors to enhance security practices.
- Methodological contribution: We demonstrate the application of Latent Dirichlet Allocation (LDA)[22] with seeded keywords for smart contract classification, offering a novel approach to analyzing blockchain data.
- Public dataset: We provide a curated dataset of categorized smart contracts, facilitating further research and analysis in the field.

Our primary aim is to conduct a detailed topic analysis on a large dataset of Ethereum smart contracts, categorising them effectively. By linking these categories to potential vulnerabilities, the research provides a foundational understanding of the challenges developers and auditors might encounter. This analysis not only highlight the diverse applications and inherent risks associated with smart contracts but also seeks to enhance their security and reliability by offering insights into common vulnerabilities. By systematically linking contract types to specific vulnerabilities, it equips developers and auditors with insights to preemptively address and mitigate risks.

Topic modeling[23] is a statistical method for uncovering the abstract topics that occur in a collection of documents. It is particularly useful in understanding large volumes of text by identifying patterns of word usage across documents. In the context of smart contracts, let us imagine to have thousands of smart contracts each with embedded comments or documentation describing their purpose and functionality. Topic modeling is able to analyse these texts to categorize smart contracts into distinct themes like "token sales," "voting systems," or "decentralised finance." It might also find that contracts with terms like "vote," "election," and "ballot" frequently occur together, suggesting a distinct topic related to voting mechanisms. This categorisation helps in understanding the common functionalities smart contracts are being used for, as well as identifying trends and potential areas of concern, such as susceptibility to certain types of vulnerabilities within a category. Latent Dirichlet Allocation (LDA)[22] is a popular topic modeling technique. It assumes each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics. LDA is particularly useful for classifying texts[24–26] in a dataset into topics in an unsupervised manner, making it ideal for analyzing smart contracts. One of the most common uses of topic modeling is text mining. However, it could be helpful in other application domains to enhance researchers' ability to interpret biological information[27], to map topics to network in networks regularization, and to gather scientific publications into several clusters[28]. Topic modeling combined with other approaches allows solving complex problems such as improving software traceability[29] in software engineering or being able to automate SC coding in certain domains[30]. Different exciting methods consist of topic modeling and clustering integration to achieve the best performance.

As shown in Fig. 1, our research model began with the collection of a corpus of Smart Contracts, followed by the application of NLP techniques to define the SC taxonomy.

- First, we have compiled, processed, cleaned, and refined public datasets to create a comprehensive dataset of 100,040 smart contracts.

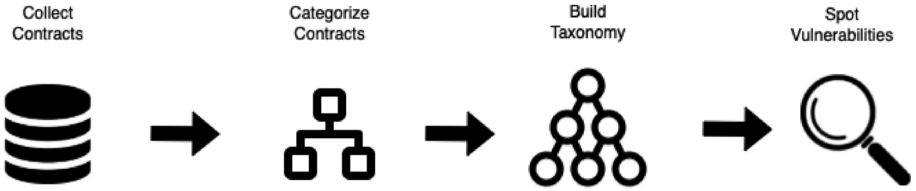


Fig. 1. Experimental design.

- Second, we have developed a refined taxonomy of smart contracts, categorizing them by their application domains.
- Lastly, we applied a combination of statistical methodologies to identify the most commonly used categories of smart contracts and how this relate to well known vulnerabilities. This enhances our understanding of Ethereum blockchain technology.

Smart contract vulnerabilities[31] refer to weaknesses or flaws within the contract code that can be exploited to cause unintended behavior, such as financial losses[32] or data breaches. These vulnerabilities arise from programming errors[33], logic mistakes, or oversight in security practices. Identifying and mitigating these vulnerabilities is crucial for maintaining the integrity and security of blockchain-based systems, as exploits can lead to significant consequences given the immutable and distributed nature of blockchain technology. We provide a detailed explanation in Section "Smart contract vulnerabilities".

The practical implications of our research are extensive, benefiting various stakeholders within the Ethereum smart contract ecosystem. For developers, our findings offer valuable categorization and detection strategies to improve the development, safety, and management of SC applications. Practitioners and investors will find the insights from this study crucial for informed decision-making, as knowledge of popular applications and trends can guide more strategic choices in the Ethereum blockchain space. For organizations and regulatory bodies, the categorizations and insights provided can inform the creation of guidelines and regulations for SC usage, potentially including the development of surveillance systems to monitor contracts and establishing domain-specific standards to augment the safety and reliability of the Ethereum blockchain ecosystem.

This paper is structured as follows: Section "Related works" reviews the background and current state of the literature. Section "Data and methods" describes our data collection and statistical methodology. The analysis results are presented in Section "Results". Section "Discussion" discusses the implications of our findings. Section "Limitations" addresses the limitations of our study. Finally, Section "Conclusions and future works" concludes with the main contributions of our study and directions for future research.

Related works

The growth of smart contracts on the Ethereum blockchain has resulted in challenges such as distinguishing specific functionalities, identifying security vulnerabilities, and managing a proliferation of nearly identical contracts due to repetitive development practices. The lack of a shared standard for evaluating the functional quality of smart contracts can lead to inefficiencies during their development cycle[34]. These issues have led researchers to develop various methods for classifying smart contracts and detecting potential threats. However, existing research often addresses isolated aspects, such as transaction behavior or code patterns, without providing a unified framework that integrates these findings into a broader context of security and functionality.

Smart contracts classification

Several approaches have been proposed for classifying smart contracts, each addressing different aspects of the problem. Hu et al.[35] introduced a transaction-based classification method that analyzes behavior patterns in over 10,000 contracts, providing a means to differentiate contract types and identify anomalies. However, this approach is primarily limited to transaction behavior, offering little insight into the contract's source code or potential vulnerabilities.

To address these limitations, Tian et al.[36] developed a multi-modal classification approach using Bi-LSTM and Gaussian LDA, which integrates information from source code, comments, and account details. While this method provides a more comprehensive analysis, it can be hindered by the absence of detailed comments or the presence of obfuscated code.

In cases where source code is unavailable, Shi et al.[37] proposed a bytecode-based classification model that focuses on features extracted from the bytecode. This approach is valuable for analyzing closed-source contracts, although it may miss specific details present in the original source code and comments.

Beyond classification, several studies have focused on vulnerability detection within smart contracts. Ferreira et al.[38] developed a tool for real-time protection against exploits, while Dingman et al.[39] presented formal classifications of known bugs. Camino et al.[40] utilized a data science approach to detect honeypots. These studies highlight the importance of security but often do not provide a broader context on the functionality and application domains of the contracts.

Recent research has further explored the potential of deep learning techniques in blockchain analysis. For instance, studies on assessing blockchain oracle reliability[41] and improving transaction provenance tracking[42] demonstrate the utility of advanced machine learning approaches. However, these studies typically focus on specific use cases rather than offering a unified taxonomy of smart contracts.

While these diverse approaches contribute valuable insights into smart contract analysis, there remains a need for a more integrated framework that encompasses both functional and security aspects, providing a holistic view of the smart contract ecosystem.

Smart contracts vulnerabilities detection

Vulnerability detection in smart contracts has been explored through various methodologies, each addressing specific challenges and limitations. Liu et al.[43] introduced a code transformation approach to detect reentrancy bugs, a common vulnerability in smart contracts. While this method effectively identifies certain types of issues, it may not cover the entire spectrum of potential security flaws inherent in smart contracts.

Another approach involves fuzzing techniques, as exemplified by ContractFuzzer[44], which employs test oracles and Ethereum Virtual Machine (EVM) logs to uncover vulnerabilities. Fuzzing can be a powerful tool for detecting known vulnerabilities; however, it often struggles with coverage limitations and may overlook more subtle or complex vulnerabilities that do not trigger during testing.

The application of deep learning to vulnerability detection, as demonstrated by Tang et al.[45] with their Lightning Cat model, offers a more modern approach. This model uses optimized deep learning techniques to detect vulnerabilities with promising accuracy. Nevertheless, deep learning models may face challenges with newly emerging vulnerabilities not present in the training data, limiting their adaptability to novel security threats[46–48].

Despite these advances, several limitations persist in the field. Many studies, including those mentioned above, are based on relatively small datasets, typically comprising 10,000 to 20,000 contracts, which may not fully represent the diversity and complexity of smart contracts on the Ethereum blockchain. Additionally, most approaches tend to focus on specific types of contracts or vulnerabilities, which can lead to a narrow understanding of the broader ecosystem.

The rapid evolution of smart contract technologies and design patterns also poses a challenge, as highlighted by Bartoletti et al.[49], whose earlier work may now be outdated. This highlights the necessity for ongoing research to keep pace with technological advancements.

Finally, existing research often treats classification and vulnerability detection as distinct areas, potentially missing valuable insights that could be gained from integrating these perspectives. Understanding the correlation between contract types and specific vulnerabilities could enhance both classification and security strategies, providing a more holistic approach to managing smart contract risks.

A recent effort to address these limitations is OpenSCV, proposed by Vidal et al.[50]. This open hierarchical taxonomy covers 94 vulnerabilities, mapped from an analysis of 77 vulnerability detection tools and research papers. The authors validate their taxonomy through expert evaluation and demonstrate its coverage of current vulnerability detection capabilities. However, while OpenSCV focuses primarily on vulnerabilities, our work aims to provide a broader classification of smart contracts based on their functionalities and purposes, in addition to considering security aspects. Our approach utilizes topic modeling techniques to automatically derive categories, offering more adaptability to emerging contract types and applications.

Contribution of the work

To provide a clear overview of the existing work and highlight the gaps in the current literature, we present a comparative analysis of key studies in Table 1, which reveals several key gaps in the literature.

Limited Dataset Size: Most studies utilize relatively small datasets, typically ranging from 10,000 to 20,000 contracts, which may not fully capture the diversity and complexity of the Ethereum smart contract ecosystem.

Lack of Complete Taxonomy: While some studies provide classification methods, few offer a comprehensive taxonomy that reflects the current state and diversity of smart contracts.

Study	Classification method	Dataset size	Vulnerability detection	SCs taxonomy	SCs category-vulnerability correlation	Time period
Hu et al.[35]	LSTM	10,000+ contracts	Limited	No	No	Not specified
Tian et al.[36]	Bi-LSTM with attention mechanism	15,213 contracts	No	No	No	Not specified
Shi et al.[37]	Bytecode analysis and machine learning	11,593 contracts	No	No	No	Not specified
Ferreira Torres et al.[38]	Transaction analysis	Not specified	No	No	No	Not specified
Dingman et al.[39]	Manual analysis and creation of Bugs Framework	Not specified	No	Partial	No	Not specified
Camino et al.[40]	Data science approach	158,863	Honeypots only	No	No	Not specified
Liu et al.[43]	Fuzzing based techniques	Not specified	Reentrancy only	No	No	Not specified
Jiang et al.[44]	Fuzzing based techniques	9,960 contracts	Yes	Yes	No	Not specified
Tang et al.[45]	Deep learning (CodeBERT, LSTM, CNN)	9,369 contracts	Yes	No	No	Not specified
Bartoletti et al.[49]	Manual analysis	834 contracts	No	Yes	No	2013-2016
Vidal et al.[50]	Hierarchical taxonomy	not specified	Yes	No	No	Not specified
Our study	Topic modeling (LDA)	100,000+ contracts	Yes	Yes	Yes	Up to 2024

Table 1. Comparison of related works in smart contract analysis.

Disconnect between Classification and Vulnerability Detection: Existing research tends to focus on either classification or vulnerability detection, but rarely combines these aspects to provide a holistic understanding of smart contract security in relation to contract types.

Some influential works, such as Bartoletti et al.[49], are based on data from the early years of Ethereum (2013-2016), and may not reflect the current state of smart contract development and usage.

Absence of Category-Vulnerability Correlation: To the best of our knowledge, no existing study explores the correlation between smart contract categories and their associated vulnerabilities.

Our study addresses these gaps by:

- Analyzing a substantially larger dataset of over 100,000 smart contracts, providing a detailed view of the Ethereum ecosystem.
- Developing an up-to-date taxonomy using topic modeling techniques, reflecting the current diversity of smart contract applications.
- Integrating classification and vulnerability detection to offer insights into the relationship between contract types and security risks.
- Providing an analysis that covers the Ethereum ecosystem up to 2024, ensuring relevance to the current state of smart contract development.
- Uniquely correlating smart contract categories with specific vulnerabilities, offering valuable insights for developers and auditors in identifying and mitigating potential risks associated with different types of smart contracts.

By integrating topic modeling with an analysis of the correlation between contract types and vulnerabilities, our study offers a fresh perspective on smart contract classification and security. This approach deepens our understanding of the smart contracts development and provides guidance for developers and auditors in identifying and mitigating risks associated with various smart contract types.

Data and methods

Dataset description

Our study's dataset on smart contract taxonomy integrates three distinct collections of Solidity programs.

The primary component is the SmartBugs repository[51], which includes over 47,000 verified Solidity smart contracts. This dataset is notable for its subset of SCs labeled with vulnerabilities, a crucial aspect for our analysis. The SmartBugs dataset is split into two sections: one listing contracts without reported vulnerabilities and the other containing contracts identified as vulnerable. This division allows a complete understanding of SC security within our taxonomy.

Additionally, the SmartCorpus repository[52] contributes a wide range of Solidity programs' source code and associated metadata. This inclusion broadens our analysis with a variety of program examples and their contextual information.

The dataset also incorporates data from the SmartSanctuary repository[53], a dynamic collection of recent, verified Ethereum SCs. It includes SCs from different networks, such as Ethereum's main and test nets, and is organized by directory. With over 70,000 main net programs, SmartSanctuary is a significant part of our dataset. The merging of these datasets forms the foundation for our analysis of smart contract applications, trends, and their interactions within the Ethereum ecosystem. During the integration of these datasets, we removed duplicates, defined as contracts with identical source code and address. This process refined our dataset to 100,040 unique smart contract samples, providing a solid base for our study.

Research methodology

Our initial approach involved manually reviewing around 600 smart contracts from our dataset. This step helped to identify the types and functionalities of the contracts we were working with.

This sample is statistically significant, as the minimum sample size that guarantees a 95% confidence level with a 5% margin of error, assuming maximum variability is given by: $n = \frac{N \cdot (Z^2 \cdot p \cdot (1-p))}{(d^2 \cdot (N-1)) + (Z^2 \cdot p \cdot (1-p))}$, where N is the population size (100,000), Z is the Z-score (1.96 for a 95% confidence level), p is the estimated proportion of an attribute that is present in the population (0.5 used for maximum variability), and d is the margin of error (0.05 for 5%). The calculation yields a minimum sample size of approximately 383 smart contracts, therefore, with a sample of 600 smart contracts, we ensure the representativeness of the broader population of smart contracts.

In this review, we found a range of categories such as Tokens, Non-Fungible Tokens (NFTs)[54], Token Exchanges, Timelock Contracts, Crowdsales, Initial Coin Offerings (ICOs), and Gambling Games. These categories are indicative of the broad spectrum of applications for smart contracts on the Ethereum blockchain.

We also encountered contracts that were either too basic or designed for specific purposes, like "Hello World" programs, which are more educational than practical. While these do not directly contribute to our taxonomy, they do offer insights into the developmental trends within the Ethereum community. Our review identified contracts associated with fraudulent activities, particularly those akin to Ponzi schemes[55]. Although these are a negative aspect of the smart contract environment, they are crucial for understanding the security challenges and for developing safer smart contract systems in the future. This initial examination laid the groundwork for our deeper topic modeling analysis, providing a clear understanding of the smart contracts in our dataset and their diverse applications. Figure 2 illustrates the research methodology, which consists of three main steps: i) Dataset construction, ii) Semantic Taxonomy construction, and iii) Vulnerability Analysis.

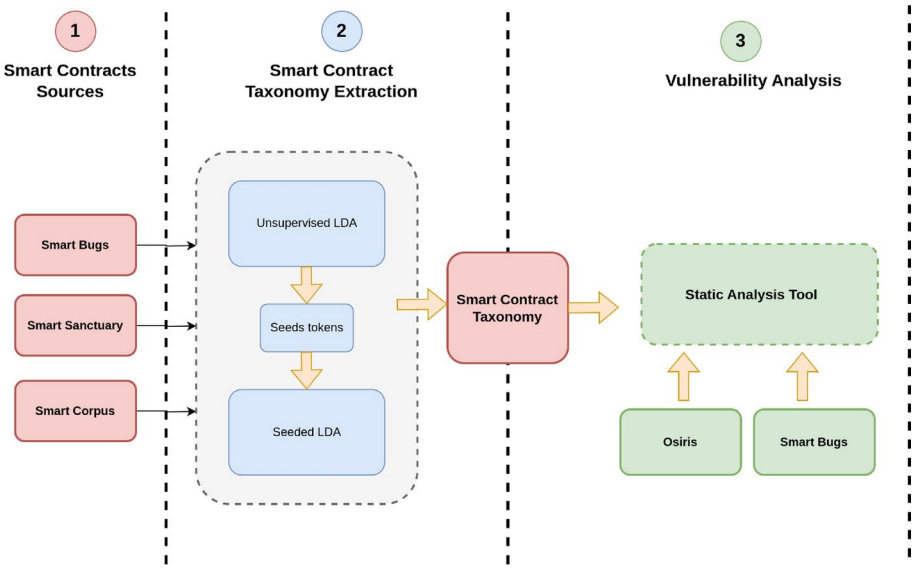


Fig. 2. Steps of the Research Methodology.

Data preprocessing and LDA model configuration

Our data preprocessing phase was essential for preparing the smart contract source code for analysis. We started by tokenizing the Solidity code, breaking it down into individual words and symbols while ensuring that significant programming constructs were maintained. This step involved handling of Solidity-specific syntax, such as contract and function declarations. We then removed common stop words and programming keywords that do not contribute to the functional categorization of the contracts. Additionally, we addressed the naming conventions in Solidity, like camelCase and snake_case, by splitting these into distinct tokens. To further streamline the data and group similar terms, we applied stemming to reduce words to their root forms. This approach was particularly important for managing variations in terminology across different smart contracts. After preprocessing, we used Term Frequency-Inverse Document Frequency (TF-IDF) vectorization to transform the tokenized smart contracts into numerical feature vectors. TF-IDF was selected for its effectiveness in emphasizing terms that are significant to a specific contract while reducing the impact of terms that are common across many contracts. To reduce computational complexity and concentrate on the most informative features, we selected the top N features based on their TF-IDF scores. The value of N was determined through experimentation to achieve a balance between retaining valuable information and ensuring model efficiency. This feature extraction process enabled us to encapsulate the core functionality of each smart contract in a format suitable for analysis with our LDA model. Configuring the Latent Dirichlet Allocation (LDA) model was a key step in our methodology. We began by determining the optimal number of topics through a systematic analysis of topic coherence scores across a range of potential topic numbers. This process involved training multiple LDA models with varying numbers of topics and evaluating their coherence to find an appropriate balance between specificity and generalizability. We then fine-tuned the model's hyperparameters which influence the distribution of topics per document and words per topic. To integrate domain knowledge and guide the topic modeling process, we carefully selected seed words for each anticipated category based on our expertise in smart contract functionalities. This seeded LDA approach allowed us to apply our prior understanding of the smart contract ecosystem while still permitting the discovery of emergent patterns in the data. We selected the LDA model for analyzing smart contracts due to its strengths in unsupervised topic discovery and its capacity to manage large corpora of text data. LDA is particularly well-suited for this task for several reasons. Firstly, its flexibility in uncovering latent themes aligns with the multi-faceted nature of smart contracts, which often serve various purposes and include different functional elements. The model's capability to assign multiple topics to a single document mirrors this complexity. Secondly, LDA offers interpretability; the topics it generates are distributions over words, which facilitate an intuitive understanding of the functional aspects of smart contracts. This feature is essential for interpreting the diverse functionalities embedded within the contracts. Furthermore, the scalability of LDA was a significant factor, given our dataset comprised over 100,000 contracts. The model's efficiency in handling large datasets ensured a practical and comprehensive analysis. Additionally, we leveraged domain knowledge through a seeded LDA approach, which allowed us to guide the topic discovery process with our initial manual insights, effectively combining unsupervised learning with domain expertise. LDA's robustness in handling sparse data was another critical consideration, as smart contract code often includes specialized terminology and unique structures. This robustness enables effective analysis despite the rarity of certain terms. Lastly, the unsupervised nature of LDA is invaluable for discovering hidden patterns and emerging categories within smart contracts.

Classification process

Our classification process involved applying the configured LDA model to the preprocessed and vectorized smart contract dataset of 100,000 contracts. Each contract was assigned probability distributions across the identified topics, and we classified each contract into the category corresponding to the topic with the highest probability. Acknowledging that smart contracts can serve multiple purposes, we also recorded secondary and tertiary classifications for contracts with significant probabilities across multiple topics. To improve our classifications, we implemented a post-processing step to address edge cases and consolidate closely related categories. This step included a manual review by three authors of this paper of the application categories extracted from the topic modeling. To validate and refine our classification system, we manually reviewed a representative sample of 600 smart contracts, representing 0.6% of the total dataset. This manual review was conducted by three authors to assess the accuracy and consistency of the unsupervised classifications and served as a baseline for evaluating the model's performance. The sample size was determined to be sufficient for initial model validation (see Section "Research methodology"). The remaining 99,400 contracts were classified using the final model developed and refined based on insights from the manual review. To ensure consistency in manual classifications, we calculated inter-rater reliability using Cohen's kappa, achieving a score of 0.76. This score indicates substantial agreement between the reviewers and confirms the reliability of the manual classification process. Based on the manual review results and the inter-rater reliability scores, we iteratively refined our classification using the LDA model. Adjustments were made to seed words, feature selection, and the number of topics to improve the model's performance and accuracy. This iterative process continued until we achieved a high level of agreement between our model and the manual reviews, resulting in a robust and reliable classification system for Ethereum smart contracts.

Smart contract topic modeling

The integration of the three datasets resulted in a CSV file capturing four essential features of Smart Contracts (SCs): address, source code, deploy date, and, where applicable, vulnerability exposure.

Our objective is to classify the SC samples in our dataset, using the source codes and comments by developers. A key observation was that many developers repurpose the source code of existing contracts on the blockchain, leading to a significant number of programs sharing similar logical structures.

Developers frequently utilize descriptive names for programming constructs such as variables, functions, structs, and contract names within smart contracts. This practice aids significantly in deciphering the purpose and functionality of these contracts. Recognizing the potential of these descriptive elements, we decided to use Natural Language Processing (NLP). Specifically, we adopted a semi-supervised approach using seeded Latent Dirichlet Allocation (LDA)[22]. Unlike standard LDA[56], which autonomously identifies topics within a document corpus, seeded LDA commences with a set of pre-defined 'seed' words for each topic. These seeds guide the topic modeling process more effectively, allowing it to align closely with established domain knowledge, a valuable asset given our initial understanding of potential topics within smart contract programming. Our use of LDA for smart contract analysis goes beyond mere statistical description. While traditional classification methods[25,26] often rely on predefined categories, our topic modeling approach allows for a more nuanced and data-driven categorization. LDA uncovers latent themes[57] within the smart contract source code, effectively classifying contracts based on their dominant topics. This method offers several advantages: (1) it allows for the emergence of categories that may not have been predetermined, potentially revealing new or evolving patterns in smart contract development; (2) it provides a probabilistic distribution of topics for each contract, offering insights into multi-purpose or complex contracts that might not fit neatly into a single category; and (3) by basing our classification on the actual content and structure of the smart contracts, we minimize biases that might be introduced by predefined categories. This approach not only classifies smart contracts[45] but also provides deeper insights into their purposes, functionalities, and potential vulnerabilities. The subsequent statistical analysis of these topic-based classifications, particularly in relation to vulnerabilities as detailed in Section "Vulnerabilities analysis", offers a complete understanding of the Ethereum smart contract ecosystem that goes beyond simple categorization. By linking these emergent categories to specific vulnerabilities, we provide a novel perspective on the security implications of different types of smart contracts, which is crucial for developers, auditors, and researchers in the field.

A critical aspect of our approach was the acknowledgment of the unique nature of smart contracts as coded entities, developed in the Solidity programming language, rather than conventional documents composed in natural language. This understanding led us to filter out these terms ensuring that frequently occurring programming terms did not skew the results. Additionally, Solidity's coding styles, specifically snake-case and camel-case, necessitated tailored preprocessing. In our case, this meant adapting our text analysis techniques to accurately parse and interpret words that were either concatenated (camel-case) or separated by underscores (snake-case).

Finally, to ascertain the most effective number of topics for our analysis, we employed the metric of topic coherence. This measure is crucial for distinguishing semantically meaningful topics from those that might simply be statistical artifacts. By evaluating various LDA models, each configured with a different number of topics, and measuring their coherence scores, we were able to select the most interpretable model. This model effectively balanced the granularity and relevance of the topics, ensuring that our analysis yielded insightful and interpretable results about the smart contracts in our dataset.

Smart contract vulnerabilities

In our exploration of smart contract vulnerabilities[4,58], we initially focused on the labelled samples from the Smart Bugs dataset[53]. However, this dataset presented a limitation: only a small fraction of contracts (about 150 out of 47,000) were linked to their corresponding exposure. To augment our understanding beyond these

labeled samples, we conducted a manual review of additional contracts. It is important to note that this manual inspection did not significantly expand our pool of labeled contracts due to the impracticality of manually reviewing a large number of Solidity contracts. To enhance the representativeness of our dataset, we employed existing tools designed for detecting vulnerabilities.

Those we considered are as follows:

- **Time Manipulation (TM)**:
  - **Explanation**: This vulnerability arises from the ability of miners to slightly alter the timestamp of a block they mine. Ethereum permits this flexibility to account for differences in system clocks and network delays. However, this can be problematic when smart contracts depend on precise timestamps for critical functions.
  - **Justification**: Time-based operations in smart contracts, such as auctions, lotteries, or timed access control, can be manipulated if they rely heavily on block timestamps. The integrity and fairness of these operations are compromised when miners can influence the timing to their advantage.
  - **Example**: Consider an auction contract that finalizes and transfers ownership of an asset to the highest bidder at a specific timestamp. If a miner is also participating in the auction and currently the highest bidder, they might manipulate the block timestamp to close the auction early, thus preventing further bids. This manipulation ensures they secure the asset, exploiting the timestamp dependency for personal gain.
  - **Mitigation**: To mitigate such vulnerabilities, smart contracts should avoid sole reliance on block timestamps for time-sensitive operations. Developers can use external oracles to obtain more accurate time data or employ block numbers as a proxy for time, though each method has its own trade-offs. These strategies help reduce the risk of time manipulation, promoting fair and secure contract behavior.
- **Arithmetic Overflow/Underflow (A)**:
  - **Explanation**: This vulnerability occurs when arithmetic operations exceed the bounds of the data type used in smart contracts. Specifically, an overflow happens when a calculation exceeds the maximum value a data type can hold, causing it to wrap around to a minimum value. Conversely, an underflow occurs when a calculation falls below the minimum value, causing it to wrap around to a maximum value.
  - **Justification**: Such vulnerabilities can lead to significant errors in smart contracts, particularly in financial calculations or any logic relying on precise numerical values. They can be exploited to manipulate contract behavior, leading to unintended consequences like incorrect balances or unauthorized token issuance.
  - **Example**: Consider a decentralized voting system where each vote is counted by incrementing a variable. If the votes are stored in a 'uint8' variable, which can hold values from 0 to 255, an overflow would occur if the vote count exceeds 255, resetting the count to 0. This could be exploited by an attacker to manipulate the election outcome, causing legitimate votes to be invalidated or miscounted.
  - **Mitigation**: To prevent overflow and underflow vulnerabilities, developers should:
    - Use the latest Solidity version, which includes built-in checks that revert transactions if an overflow or underflow is detected.
    - Conduct thorough testing and auditing of smart contracts to identify potential overflow/underflow issues.
    - Implement safe arithmetic libraries, such as OpenZeppelin's SafeMath, which provide functions that automatically handle these conditions.
- **Bad Randomness (BR)**:
  - **Explanation**: In blockchain systems, achieving true randomness is challenging due to the deterministic nature of the network. Nodes must reach a consensus on the same data, making it difficult to implement unpredictability directly on-chain. Typically, randomness is derived from block-related data such as hashes, block numbers, gas limits, or timestamps. However, these methods are susceptible to manipulation, particularly by miners who have some control over these values.
  - **Justification**: Randomness is critical in various smart contract applications, such as lotteries, games, or any process requiring unbiased random selection. If the randomness can be influenced, it compromises the integrity and fairness of these applications. The predictability of supposedly random outcomes can lead to significant exploits, undermining user trust and the security of the blockchain ecosystem.
  - **Example**: Consider a decentralized lottery system that uses the hash of the most recent block as the source of randomness. Since miners can influence the block hash, they could potentially manipulate the timing of block creation to achieve a favorable outcome, thereby skewing the results of the lottery. This manipulation not only affects fairness but also presents a risk of financial loss to other participants.
  - **Mitigation**: To mitigate the risks associated with bad randomness, developers should:
    - Utilize external oracles that provide verifiable random data from off-chain sources, ensuring the randomness is less susceptible to manipulation by on-chain actors.
    - Implement commit-reveal schemes, where participants first commit to a value and later reveal a random input, which is combined to generate a final random number. This reduces the potential for any single participant to control the outcome.
    - Consider using future block attributes, such as block hashes from future blocks, which are harder for miners to predict and manipulate. These strategies help ensure that the generated randomness is as

secure and unpredictable as possible, thereby enhancing the fairness and security of smart contract applications.

- **Unchecked Low-Level Calls (ULLC)**:
  - **Explanation**: This vulnerability arises from the use of low-level functions in Solidity, such as call, delegatecall, or callcode. These functions provide significant flexibility by allowing direct interactions with other contracts or addresses, including sending Ether and executing code. However, they also bypass certain safety features provided by higher-level functions.
  - **Justification**: The primary risk of using low-level calls is that they do not automatically check if the called function was successful. If the return value of these calls is not checked, it can lead to unexpected behaviors, such as failing to revert the state of a contract after an unsuccessful operation. This oversight can be exploited by malicious contracts, potentially leading to financial losses or unauthorized actions within the contract.
  - **Example**: Consider a contract that uses the call function to send Ether to another address. If the call fails (e.g., due to insufficient gas or a failure in the receiving contract), and the calling contract does not check the return value, the Ether might not be transferred, but the contract could still assume the operation was successful. This scenario could lead to inconsistencies in the contract's state or loss of funds. An attacker might also create a contract that deliberately fails or consumes excessive gas, exploiting the lack of error handling in the caller contract.
  - **Mitigation**: To prevent such vulnerabilities, developers should:
    - Prefer using high-level functions, such as transfer and send, which automatically handle error checks and limit gas forwarding.
    - Always check the return value of low-level calls to ensure the operation was successful. If the call fails, appropriate fallback mechanisms should be in place to handle the failure.
    - Implement the checks-effects-interactions pattern, which helps prevent reentrancy attacks by ensuring that all state changes occur before interacting with external contracts. By following these practices, developers can enhance the security and robustness of smart contracts, minimizing the risks associated with unchecked low-level calls.

- **Access Control (AC)**:
  - **Explanation**: Access control refers to the security mechanisms that determine who is allowed to perform specific actions within a system. In smart contracts, access control mechanisms are used to specify which addresses can call certain functions, particularly those that handle sensitive operations such as transferring funds, altering contract parameters, or managing user roles.
  - **Justification**: Properly implemented access control is crucial for the security of smart contracts. If access control mechanisms are weak or improperly configured, unauthorized users may gain the ability to execute critical functions. This can lead to unauthorized asset transfers, changes to contract logic, or even complete takeover of the contract. Therefore, robust access control mechanisms are essential to prevent unauthorized actions that could compromise the contract's integrity and security.
  - **Example**: A smart contract may have a function for withdrawing funds or transferring ownership of assets. If the contract does not properly verify the caller's authorization before executing these functions, an attacker could exploit this vulnerability to withdraw funds or change ownership without permission. For example, a missing or improperly implemented modifier that checks for an authorized user could allow any caller to execute the function, leading to significant financial loss or unauthorized control over the contract.
  - **Mitigation**: To mitigate access control vulnerabilities, developers should:
    - Use modifiers in Solidity to enforce access restrictions. Modifiers can check whether the caller has the necessary permissions before allowing access to sensitive functions.
    - Clearly define roles and permissions within the smart contract, ensuring that only authorized addresses can perform critical actions.
    - Regularly audit the access control logic to identify and fix any potential gaps or weaknesses.
    - Implement multi-signature wallets for high-value actions, requiring multiple approvals before executing significant transactions or changes. By implementing these measures, developers can help ensure that only authorized parties can execute sensitive operations, thereby protecting the contract and its users from unauthorized access and manipulation.

- **Concurrency (C)**:
  - **Explanation**: Concurrency vulnerabilities occur when the order of transactions influences the outcome of a smart contract's operations. On the Ethereum blockchain, miners have the discretion to order transactions within a block, which can be exploited by malicious actors to alter the expected sequence of events. This is particularly problematic in scenarios where transaction timing can affect the state or result of the contract.
  - **Justification**: In applications like decentralized finance (DeFi), where financial transactions are common, concurrency issues can lead to significant financial manipulation. The most common manifestation of this vulnerability is known as "front running," where an attacker, aware of a pending transaction, submits a

competing transaction with a higher gas fee to prioritize its inclusion in the blockchain. This allows the attacker to manipulate the market or transaction conditions to their advantage, potentially causing financial loss to other participants.
  - **Example**: In a DeFi application, if an attacker notices a large buy order for a specific cryptocurrency, they might front-run this transaction by submitting their buy order with a higher gas fee. Once their transaction is processed first, the price of the asset increases, allowing the attacker to sell it at a higher price after the original transaction goes through, profiting from the manipulated price change.
  - **Mitigation**: To mitigate concurrency vulnerabilities, developers should:
    - Implement commit-reveal schemes, where users first commit to a transaction without revealing its details, and then reveal the actual transaction information in a subsequent phase. This helps to ensure that the final actions are based on a full set of data and reduces the advantage of transaction timing.
    - Design contracts to be less dependent on transaction order. For instance, using mechanisms like time windows for accepting transactions can help ensure that the order does not affect the final outcome.
    - Incorporate randomization or other methods to obscure transaction details until after a decision point, making it harder for attackers to predict and exploit the order of transactions. By adopting these strategies, developers can enhance the resilience of smart contracts against order-based attacks and ensure fairer interaction outcomes.

- **Reentrancy (R)**:
  - **Explanation**: Reentrancy vulnerabilities occur when a smart contract makes an external call to another contract before updating its state. If the called contract has a fallback function or any method that interacts with the original contract, it can make a reentrant call back into the original contract before its initial transaction is completed. This can exploit the original contract's state in an inconsistent manner, leading to unexpected or harmful outcomes.
  - **Justification**: This type of vulnerability is particularly dangerous because it can allow attackers to execute repeated actions, such as multiple withdrawals, by recursively calling a function before the original transaction completes. This can lead to significant financial losses and has been the cause of some of the most notorious exploits in blockchain history.
  - **Example**: The most well-known example of a reentrancy attack is the DAO hack in 2016. In this incident, an attacker exploited a reentrancy vulnerability in a smart contract to repeatedly withdraw Ether. The attack was possible because the contract failed to update its internal state before transferring funds, allowing the attacker to drain millions of dollars' worth of Ether by calling the withdrawal function multiple times in quick succession.
  - **Mitigation**: To prevent reentrancy vulnerabilities, developers should:
    - Implement the Checks-Effects-Interactions pattern, where the contract first checks all conditions and updates its state before making any external calls. This ensures that the contract's state is always consistent before interacting with external entities.
    - Use reentrancy guards, such as a mutex or simple lock mechanism, to prevent multiple entries into a function until the first invocation has completed. This technique helps ensure that once a function is being executed, it cannot be re-entered until the initial execution is fully resolved.
    - Avoid writing complex logic in fallback functions, as they are often the entry points for reentrant calls. By following these best practices, developers can significantly reduce the risk of reentrancy attacks, thereby enhancing the security and reliability of their smart contracts.

- **Denial of Service (DoS)**:
  - **Explanation**: A Denial of Service (DoS) attack aims to render a smart contract inoperable, either temporarily or permanently. This is typically achieved by exploiting flaws in the contract's design, such as those that allow for excessive gas consumption or the blocking of essential functions. Such vulnerabilities can prevent the contract from processing legitimate transactions, effectively taking it offline.
  - **Justification**: DoS vulnerabilities are critical because they can make a contract unavailable to users, disrupting services, and potentially causing financial losses. In the context of blockchain, where contracts are often immutable once deployed, such vulnerabilities can have long-lasting effects if not properly addressed during the development phase. Ensuring availability and reliability is crucial for maintaining trust in decentralized applications.
  - **Example**: One common DoS attack method involves creating infinite loops or excessively large iterations within a contract. For example, if a contract includes a function that iterates over an array or mapping that can be influenced externally, an attacker might exploit this to insert large amounts of data. This can lead to out-of-gas errors, where the contract consumes all available gas during execution, thus preventing the function from completing and blocking further transactions. Another example is when access to certain functions is restricted to the contract's owner or a specific address. If an attacker gains control over this privileged account, they can block critical functions, effectively taking the contract offline.
  - **Mitigation**: To prevent DoS vulnerabilities, developers should:
    - Avoid unbounded loops and ensure that any iteration is capped or otherwise controlled, preventing excessive gas usage. Where loops are necessary, consider splitting the data processing across multiple transactions to avoid hitting gas limits.

- Implement gas-efficient algorithms and optimize contract code to minimize unnecessary gas consumption.
- Ensure rigorous access control measures, preventing unauthorized users from calling critical functions. Use multi-signature wallets for sensitive operations to distribute control and reduce the risk of a single point of failure.
- Conduct thorough testing, including stress testing under various scenarios, to identify and address potential DoS vulnerabilities before deployment. By following these practices, developers can improve the resilience of smart contracts against DoS attacks, ensuring better availability and security for users.

Results

In our study, we specifically configured the Latent Dirichlet Allocation (LDA) model to identify 15 topics within our collection of smart contracts. This decision was based aiming to strike a balance between granularity and manageability. By setting the model to uncover 15 distinct topics, we intended to capture a wide spectrum of application domains present in our corpus, ensuring a complete understanding of the smart contracts' functionalities. This approach allowed us to systematically categorise our corpus of contracts into meaningful groups that reflect their primary purposes and application areas[59]. Upon reviewing the results, we observed the potential for merging some categories. For instance, the *Crowdsale* and *ICO programs*, predominantly designed for token sales, share a similar structure with *Bid programs*, which are typically used for purchasing a variety of goods. Crowdsale and ICO (Initial Coin Offering) programs are blockchain-based fundraising mechanisms, where new projects sell their underlying crypto tokens in exchange for Bitcoin, Ethereum, or other cryptocurrencies. They are designed to raise capital for new crypto ventures and often involve pre-selling tokens to investors before the public launch. On the other hand, Bid programs are smart contracts designed for auctions or competitive bidding processes, allowing users to bid on items or services. Despite their different purposes–fundraising vs. auctioning–both types of programs use similar smart contract structures to facilitate transactions and agreements on the blockchain. Given their structural similarities, these categories were merged into one.

The effectiveness of the LDA model was increased by providing it with specific *seed* terms for each topic, effectively setting an *a priori* probability distribution for the terms within a topic.

The term *seed* in the context of LDA refers to pre-selecting specific terms to guide the topic modeling process. By introducing seed terms for each topic, we essentially provide the LDA model with a starting point or a hint about the kind of vocabulary that defines a topic, thereby setting an a priori probability distribution. This means that certain words are more likely to be associated with specific topics from the outset. This approach can significantly enhance the model's ability to accurately categorize documents according to the predefined topics by aligning the model's learning process more closely with our expectations and the unique characteristics of our dataset.

For example, terms such as *'start'*, *'end'*, *'lock'*, *'eth'*, *'duration'*, and *'unlock'* are indicative of time constraints and influenced the distribution of topic terms. When LDA model processes a document and discovers it contains specific keywords that align with the characteristics of a predetermined topic, the model then mathematically increases the probability, that this document is an example of that topic. This adjustment is based on how closely the document's content matches the thematic captured by the seed terms associated with each topic, thereby enabling a more accurate categorisation of documents within the dataset. For instance, a document with terms like *'play'*, *'player'*, *'bet'*, *'dice'*, *'pot'*, *'prize'*, etc., is likely to be categorized as a gambling game contract. Conversely, terms such as *'lock'*, *'start'*, *'end'*, *'time'*, *'deposit'*, *'deposit date'*, and so on, are closely related to Banking programs that operate within specific time constraints. Similarly, the gambling category has been expanded to include oracle smart contracts, and the investment contracts category now subsumes Ponzi schemes.

We finally defined the following eleven categories:

- **Bank** contracts implement a virtual bank, allowing participants to store their Ether. This category is strictly correlated to the Ether Lock / Time Constraints one, but the difference is that a user can withdraw his money at any moment. Often these programs are also used for token storing.
- **Bid** contracts implement auctions to buy a specific good. These contracts also include ICO and Crowdsales. These programs are also widely used for token marketing.
- **Certification and NFT** (CNFT) are contracts certifying the authenticity and the ownership of a digital asset. It is one of the most popular trends concerning cryptocurrency topic.
- **Chain Management** (CM) programs implement chain operations. This category is correlated to the Wallet one, because both performs operations to simplify the interaction with the blockchain.
- **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. These contracts provide the possibility to use tokens for shopping particular items.
- **Money Investment** (MI) 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, like burn and exchange.
- **Wallet** programs act like wallets, performing transactions, storing Ether, etc.
- **Unknown** category considers the documents unrelated to any of the categories above (for example, Hello World, nonsense operations, and useless mathematical operations programs).Each category has peculiar keywords which define the application domain as presented in Table 2.

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

Table 2. Categories and top 5 keywords per topic.

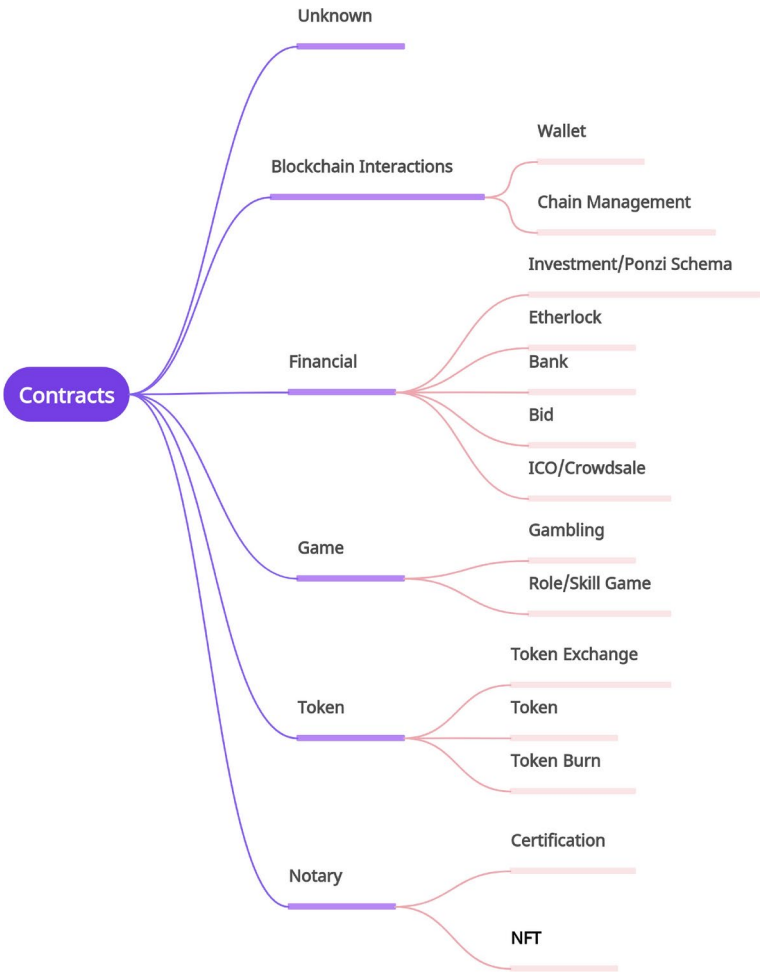


Fig. 3. Smart contracts taxonomy by macro-category.

We did not consider the Library category, since it is almost impossible that developers deploy a contract whose purpose is exclusively to act as a Library, so we did not have it as a category, even because nowadays Library contracts are integrated into other contracts.

Our taxonomy, which is resumed in Fig. 3, consists of the following macro-categories:

- **Notary**: all the programs certifying ownership or authenticity belong to the Notary category, which is the main reason why we did not consider Non-Fungible-Tokens belonging to the Token class.
- **Token**: developers can follow several patterns to build their token contracts. The most common practice is the 'simple token': these programs contain only a token structure and few tokens transfer and burn functions. However, our analysis with the seeded LDA considers all the token patterns unique.

- **Game**: consists of both gambling and role games.
- **Financial**: includes all the contracts managing or gathering money (and sometimes tokens). The included sub-categories are ICO, Crowdsale, Bid, Bank, and Ether Lock programs.
- **Blockchain** interaction: the Wallet and Chain Management categories are strictly bounded, and sometimes wallet programs could also perform operations simplifying the interaction with the blockchain.
- **Unknown**: all the contracts with undefined behavior belong to this macro-category.

Finally, Fig. 4 shows the categories of smart contracts and the number of samples per category in our dataset, after seeding the LDA model. We can see that Token and Certification and NFT are the most used domain of application.

Temporal taxonomy

Understanding the evolution of smart contract applications over time is important for grasping the dynamic nature of the Ethereum blockchain ecosystem. To this end, we developed a temporal taxonomy based on the categories identified by our seeded LDA model. This approach not only helps in categorising smart contracts but also in tracking their development and popularity trends over the years. Drawing from the taxonomy framework of Bartoletti et al. (2017)[49], we adapted and refined it to reflect the latest developments in Ethereum since its early years. Our methodology grouped the categories from the LDA analysis into broader macro-categories, offering a comprehensive view of the smart contract landscape across different time periods.

One trend observed is the dominance of token design patterns, alongside the growing prominence of Certification and NFT (CNFT) design patterns. The increasing prevalence of CNFT contracts, particularly after the launch of Cryptokitties in November 2017, underscores the evolving interests and applications within the Ethereum community. A decline in Token samples from 2019 was noted, attributable to the smaller dataset for that year.

Figure 5 illustrates the evolving trends in smart contract (SC) categories from 2017 to 2021. Initially, categories such as "Bank," "Bid," and "Game" demonstrated modest activity, reflecting the early experimental phase of smart contract applications. This period was characterized by a cautious exploration of the potential uses of blockchain technology in financial services, auctions, and gaming. The graph indicates a notable increase in the "Certification and NFT" (Non-Fungible Tokens) category, particularly peaking in 2021. This surge corresponds with the growing interest in digital ownership and asset tokenization, where NFTs have become a central element in representing unique digital assets, including art, collectibles, and intellectual property. The expansion of NFTs beyond digital art into other areas such as virtual real estate and intellectual property rights highlights their versatility and increasing importance in the blockchain ecosystem. The "Token" category, encompassing utility and security tokens, shows robust and consistent growth throughout the period. This trend underscores the essential role of tokens in the crypto economy, facilitating a wide range of functionalities from fundraising (e.g., ICOs) to access rights within decentralized applications. Additionally, the "Unknown" category, though relatively small at 0.28%, points to the emergence of new and potentially innovative domains within the smart contract landscape that have not yet been fully defined or categorized. This minor yet significant presence suggests a continuous evolution in the types of applications being developed, highlighting the dynamic and ever-expanding

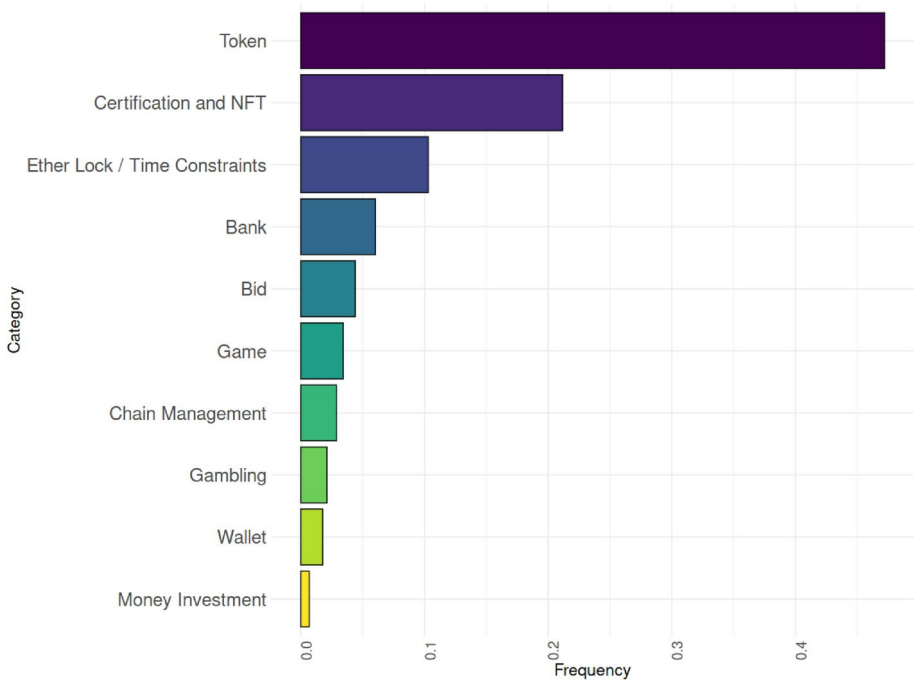


Fig. 4. Smart contracts categories distribution.

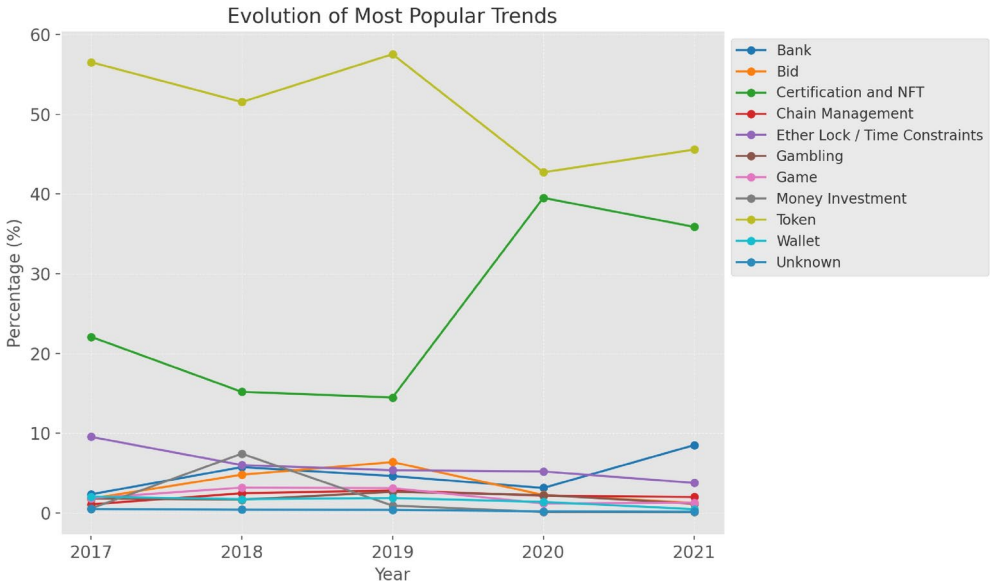


Fig. 5. Most popular trends over ethereum lifecycle.

	Access control	Arithmetic	Bad randomness	Concurrency	DOS	Reentrancy	Time manipulation	Unchecked LLC
Bank	2	130	0	11	2	1	0	7
Bid	0	77	0	24	0	1	1	14
CNFT	0	285	0	83	0	8	22	18
CM	0	53	0	1	0	0	0	10
ELTC	0	133	0	10	0	17	21	12
Gambling	0	29	4	3	0	2	2	15
Game	0	45	0	4	0	1	1	8
MI	0	15	0	1	0	1	2	1
Token	0	1228	0	62	0	4	77	12
Wallet	0	39	0	1	0	0	0	3

Table 3. Contingency table of categories and vulnerabilities.

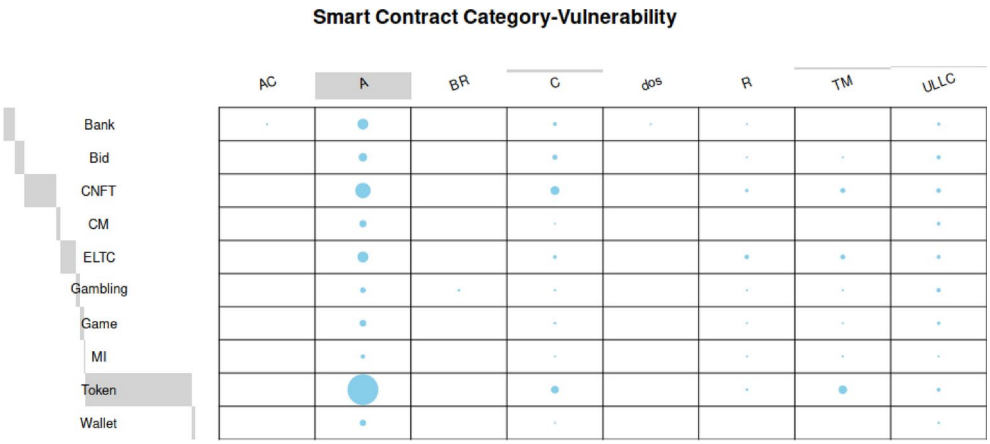
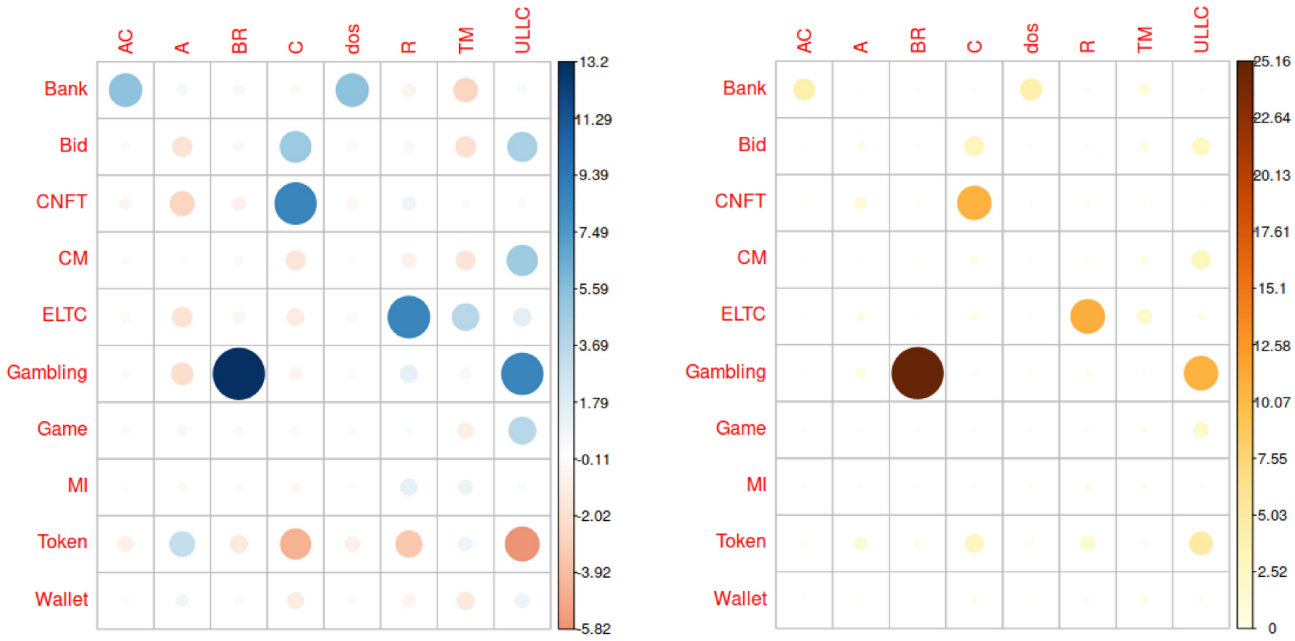


Fig. 6. Graphical representation of the category vs single vulnerability contingency table.

	Access control	Arithmetic	Bad randomness	Concurrency	DOS	Reentrancy	Time manipulation	Unchecked LLC
Bank	5.37	0.51	− 0.49	− 0.35	5.37	− 0.78	− 2.78	0.36
Bid	− 0.31	− 1.85	− 0.43	4.79	− 0.31	− 0.50	− 2.01	4.31
CNFT	− 0.58	− 2.89	− 0.82	8.63	− 0.58	0.91	0.23	0.34
CM	− 0.23	0.14	− 0.32	− 1.82	− 0.23	− 0.95	− 1.79	4.65
ELTC	− 0.39	− 1.90	− 0.56	− 1.38	− 0.39	8.71	3.62	1.54
Gambling	− 0.21	− 2.35	13.20	− 0.67	− 0.21	1.40	− 0.46	8.64
Game	− 0.22	− 0.43	− 0.31	− 0.33	− 0.22	0.19	− 1.14	3.68
MI	− 0.13	− 0.31	− 0.18	− 0.47	− 0.13	1.36	0.99	0.22
Token	− 1.05	3.11	− 1.49	− 4.61	− 1.05	− 3.49	0.88	− 5.82
Wallet	− 0.19	0.69	− 0.26	− 1.31	− 0.19	− 0.78	− 1.47	0.98

Table 4. Chi-squared residuals.



(a) Residual correction plot single vulnerability. **(b)** Contribution of each cell to the total χ^2.

Fig. 7. Residual and Contribution Plot for Single Vulnerability.

nature of the blockchain ecosystem. Overall, these trends illustrate a broader shift in how smart contracts are utilized, moving from niche experiments to more complex and diverse applications. This evolution reflects the growing maturity of blockchain technology and its adoption across different sectors, indicating a trend towards more specialized and sophisticated uses of smart contracts. The consistent growth in key categories like "Token" and the rise of NFTs suggest that these technologies will continue to play a pivotal role in the future of digital economies.

Vulnerabilities analysis

The analysis of vulnerabilities in smart contracts is a critical aspect of ensuring the security and integrity of the Ethereum blockchain. Identifying and understanding these vulnerabilities is key for developers and stakeholders to mitigate risks and enhance the reliability of smart contract applications. With this in mind, our study utilized the Osiris tool[60] for our vulnerabilities detection analysis. Osiris was selected for its accuracy in identifying key vulnerabilities such as arithmetic bugs, time manipulation, reentrancy, concurrency, and low-level calls. While there are other tools available, such as ContractFuzzer and Slither, each tool has its own focus and limitations. Slither, for example, excels in identifying vulnerabilities related to poor Solidity programming practices, but it does not align with our broader vulnerability detection objectives. ContractFuzzer offers detection of reentrancy, time manipulation, and low-level calls, similar to Osiris, but it also requires additional files like the contract's ABI and bin files, and deployment on a private chain. Furthermore, ContractFuzzer does not include arithmetic bug detection, which was a significant factor for us given the presence of arithmetic bugs in our subset of labeled contracts.

Osiris operates by first compiling the contract and then analyzing it for vulnerabilities, reporting all detected issues in a log. Using Osiris, we were able to identify eight different categories of vulnerabilities in our dataset, namely Ether Lock / Time Constraints (ELTC), Gambling, Game, Token, Wallet, Bank, Chain Management (CM), and Certification and NFT (CNFT). The specific vulnerabilities we focused on were Time Manipulation (TM), Arithmetic (A), Bad Randomness (BR), Unchecked Low-Level Calls (ULLC), Access Control (AC), Denial Of Service (DOS), Concurrency (C), and Reentrancy (R).

However, we encountered a significant challenge: Osiris could not compile a large portion of the Solidity programs in our dataset. This issue likely stemmed from the compiler version bundled with the Osiris Docker container, which is outdated compared to the newer Solidity pragmas in many of our contracts. Upgrading the compiler without modifying the tool could lead to additional issues, making this approach impractical.

Despite this limitation, our analysis covered 3114 vulnerable smart contracts, providing valuable insights into the security landscape of Ethereum smart contracts.

Table 3 presents a contingency table showing the relationship between smart contract categories and their vulnerabilities. Figure 6 visually depicts this table, using circles of varying sises within each cell to represent the strength of association between specific categories and vulnerabilities. The larger the circle, the more frequent the association. A gray bar alongside each row or column indicates the overall frequency of each category or vulnerability, providing a baseline for comparison. The pattern observed suggests a non-random relationship between categories and vulnerabilities, highlighting specific areas where vulnerabilities are more prevalent, indicating a significant link between the nature of a smart contract's category and its susceptibility to certain vulnerabilities.

To test the independence hypothesis, we used a Chi-square test in order to examine whether Categories and Vulnerabilities (rows and columns of the contingency table) are statistically independent. For this test, the null and alternative hypothesis are:

- H_0: all the *observed* joint frequencies of Category and Vulnerability cells are equal to the *expected* joint frequencies.
- H_1: there are *observed* joint frequencies of Category and Vulnerability cells that differ from the *expected* joint frequencies.

For each cell of the contingency table, we have to calculate the expected value under the null hypothesis. For a given cell, the expected value of the joint frequency is calculated as follows: $e_{i,j} = \frac{\sum_1^k row_i * \sum_1^c col_j}{\#\ observations}$. The Chi-Square statistic is calculated as: $\chi^2 = \sum_{i=1}^{k} \sum_{j=1}^{c} \frac{(o_{i,j} - e_{i,j})^2}{e_{i,j}}$, where the sum is extended to all couples (Category, Vulnerability). We obtained a $\chi^2 = 131.54$ and a $p - value = 1.048e - 09$, indicating that we must reject the null hypothesis and conclude that the two variables, Category and Vulnerability, are significantly associated.

Each cell contributes to the total Chi-square score with the cell's Chi-square residual statistic: $r_{i,j} = \frac{o_{i,j} - e_{i,j}}{\sqrt{e_{i,j}}}$

, the so-called Pearson residual as shown in Table 4. Figure 7a shows the Pearson residuals, where circle size is proportional to the cell contribution. Here the sign of the standardized residual is interpreted as follows.

- Positive residuals are highlighted in blue. They indicate a positive association ("attraction") between the relevant Category (row) and Vulnerability (column).
- Negative residuals are shown in red. They indicate that the associated Category (row) and Vulnerability (column) variables have a negative relationship ("repulsion").

Figure 7a shows that the majority of all associations are positive, and it is also possible to compute the contribution of each cell by the following ratio: $contr_{i,j} = \frac{r_{i,j}^2}{\chi^2}$, where $r_{i,j}^2$ is the cell Pearson residual. 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. Figure 7b highlights the contribution of each cell, providing insights on the nature of the relationship between Categories (rows) and Vulnerabilities (columns) of the contingency table. For example, Smart Contracts belonging to the Gambling category are *strongly* associated to Bad Randomness (they contribute to 25.15% of the total χ^2), while Certification and Non-Fungible Tokens Category and Concurrency vulnerability contribute for the 12.32% of the total χ^2. ELTC and Reentrancy provide a similar contribution, the 12.31% of the total χ^2 as well as Gambling and ULLC (13.32% of the total χ^2). A smaller contribution is provided by Ether Lock/Time Constraint and Reentrancy (8.69% of the total χ^2). All of these couples are thus *frequently* associated to each others.

Discussion

The taxonomy revealed by our LDA model reflects the underlying patterns and commonalities within the smart contract corpus, shaped by developers' collective priorities, regulatory influences, and market demands. The emergence of distinct categories such as Financial, Game, Token, and Notary is indicative of the principal sectors where Ethereum's technology has been most extensively applied. The Financial category, with its subdivisions like Bank, Bid, ICO/Crowdsale, and even Investment/Ponzi Schemes, mirrors the blockchain's significant impact on financial transactions and its potential for both legitimate and illicit economic activities. This could be attributed to Ethereum's inherent features that facilitate transparency, programmability, and trustless exchanges, which are cornerstones of financial applications. The clear demarcation between Gambling and Role/Skill-based Games suggests a maturation in the market for decentralized applications (dApps), which cater to varied user

intentions–from entertainment to competitive skill engagement. This distinction may also reflect a community response to regulatory scrutiny, as gambling contracts require strict adherence to legal frameworks across jurisdictions. The Token taxonomy–encompassing Exchange, Standard Tokens, and Token Burn–indicates the evolving sophistication in token economics and management. The presence of Token Burn contracts points to advanced strategies for managing token supply and value, a concept native to blockchain-enabled digital assets. The Notary section, with its focus on Certification and NFTs, is perhaps the most illustrative of the blockchain's non-financial applications. The rise of NFTs, in particular, points to a significant shift in how digital ownership and provenance are conceptualized and secured, propelled by broader cultural adoption. The taxonomy's structure also hints at the blockchain's shifting role from a purely transactional platform to a more complex ecosystem that encapsulates a wider range of human activities, including art, collectibles, and even identity verification. As the Ethereum blockchain continues to evolve, it is likely that this taxonomy will expand and diversify further, reflecting new innovations and the continuous reshaping of the blockchain's capabilities to meet user demands and technological advancements.

The analysis of vulnerabilities within different categories of smart contracts on the Ethereum blockchain, as revealed by our study, leads to several interpretations and hypotheses.

- **Gambling and Bad Randomness**: the strong association between smart contracts in the Gambling category and the Bad Randomness vulnerability can be attributed to the inherent challenges of implementing true randomness in blockchain environments. Given that blockchain is deterministic by nature, achieving randomness without external sources is complex. Gambling contracts often rely on pseudo-random techniques, which are vulnerable to manipulation, explaining their significant contribution to this vulnerability type.
- **Certification and Non-Fungible Tokens (CNFT) and Concurrency**: the correlation between CNFT and Concurrency vulnerabilities might stem from the operational dynamics of NFTs. These contracts often involve multiple transactions and interactions within a short time frame, making them susceptible to concurrency issues. This vulnerability is particularly critical in scenarios where the order of transactions can affect the outcome, such as in the minting and trading of NFTs.
- **Ether Lock/Time Constraint and Reentrancy**: the association between Ether Lock/Time Constraint and Reentrancy vulnerabilities could be related to the mechanisms used in contracts to lock funds for a certain period. Contracts with time-based constraints are likely to implement functions that handle deposits and withdrawals, making them potential targets for reentrancy attacks.
- **Gambling and Unchecked Low-Level Calls (ULLC)**: the connection between Gambling contracts and ULLC vulnerabilities suggests that these contracts might frequently employ low-level call functions for their operations, which if unchecked, can lead to security risks. This could be due to the complex payout mechanisms often involved in gambling contracts, which might necessitate the use of such functions.The high contributions of these vulnerabilities to the total Chi-square score indicate that they are not random occurrences but rather indicative of underlying patterns in how smart contracts are developed and the inherent risks associated with their respective categories. Furthermore, the observed positive associations (attractions) across most categories and vulnerabilities underscore the need for heightened security practices in smart contract development. It suggests that despite the evolution of the Ethereum platform and its associated tools, certain vulnerabilities remain prevalent and need to be addressed with tailored strategies depending on the contract's category.

Limitations

In our study we identify several validity threats. Construct validity is challenged by the LDA model and seeded keywords' ability to truly capture smart contract functionalities and vulnerabilities. Internal validity could be compromised by biases in selecting these keywords or configuring the LDA, affecting topic accuracy. External validity raises questions about the applicability of our findings to other blockchain platforms or contracts beyond our dataset. Lastly, reliability concerns the consistency of our model's performance in categorizing smart contracts across various instances. The application of NLP and topic modeling to smart contract analysis brings to light several challenges in selecting appropriate keywords. Algorithms like Latent Dirichlet Allocation (LDA) rely on the presence and distribution of specific keywords within the text corpus to deduce the underlying topics. This reliance necessitates the use of descriptive and consistently named keywords across documents for the analysis to be effective. However, the nature of smart contracts, often characterised by varied and sometimes non-descriptive naming conventions, can significantly hinder the topic modeling process. Inconsistencies or the use of generic terms in naming variables, functions, or even the contracts themselves may lead to inaccurate topic inference, misrepresenting the thematic structure of the dataset. This limitation is critical as it directly impacts the ability of the model to accurately categorise and analyse the smart contracts, potentially obscuring key insights into their functionalities and areas of application.

To address the challenges in keyword-based topic modeling, we enhance the LDA approach with a method known as seeded LDA. This enhancement involves integrating a handpicked set of keywords (seeds) into the model. These seeds, carefully chosen based on domain expertise and analysis, aim to infuse the topic modeling process with specific knowledge about the subject area. By doing so, seeded LDA is better positioned to direct the algorithm's focus towards generating topics that are not only relevant but also deeply reflective of the underlying themes within smart contracts. This methodological augmentation improve the precision of topic discovery, ensuring that the resulting categories are both meaningful and closely aligned with the actual content and intentions of the smart contracts under examination. This limitation suggests a trade-off between the model's targeted focus and its understanding of the domain, emphasising the need for careful seed selection and ongoing refinement to bridge any gaps in representation.

While our taxonomy provides a structured understanding of smart contract vulnerabilities and their correlation with contract types, we recognize that its immediate adoption by developers may face certain

challenges. Developers may not typically use formal academic taxonomies or classification schemes in their day-to-day workflows, which could initially limit the direct application of our findings. However, the key contribution of our taxonomy is to offer developers insights into potential security risks associated with different types of smart contracts before vulnerabilities are detected. Since developers usually know the application domain of their smart contracts in advance, this taxonomy can serve as a proactive guide. By highlighting areas where specific contract types are statistically more likely to encounter certain vulnerabilities, developers can focus their efforts on mitigating these risks early in the development process. Our taxonomy complements existing smart contract analysis tools by providing a broader context on vulnerability trends across different contract types. This added layer of understanding can help developers prioritize security measures and refine their testing strategies more effectively. To enhance its practical usability, future work could focus on integrating our taxonomy into tools and platforms already familiar to developers, such as static analysis tools, code editors, or automated testing frameworks.

Conclusions and future works

This study provides an in-depth statistical analysis of Ethereum smart contracts, categorizing them into distinct domains and tracking their evolution over time. Using Latent Dirichlet Allocation (LDA), we identified ten unique categories of smart contracts, expanding the existing literature by offering a more detailed classification. Our analysis not only delineates the predominant categories but also explores the relationship between these categories and specific vulnerabilities, highlighting the distinct risk profiles associated with each domain. The vulnerability analysis reveals a significant correlation between certain smart contract categories and particular security risks. These findings emphasize the importance of implementing targeted security measures for different types of contracts. This information is crucial for developers and auditors within the Ethereum ecosystem, providing a foundation for more precise and effective security practices. The taxonomy developed in this study enhances our understanding of the current smart contract landscape, showcasing the wide range of applications and complexities within blockchain technology. By establishing connections between contract types and specific vulnerabilities, our research offers the blockchain community valuable insights for improving the security and reliability of smart contracts. Future research could build on this work by examining how vulnerability patterns change over time across different smart contract categories and assessing the effectiveness of targeted security measures for these categories.

Data availability

The datasets generated and/or analysed during the current study are available in the Top-Trending-Contracts repository, https://github.com/giacomofi/Top-Trending-Contracts. To support ongoing research and community engagement, we have established a GitHub repository, currently featuring a collection of the top-trending contracts categorized as ELTC, Bank, CNFT, and Token programs, organized chronologically. While the repository presently focuses on these trending categories, our plan is to progressively include a broader spectrum of contracts, particularly those representing other categories and various vulnerabilities. This expansion will not only improve the repository's but also provide insights into the vulnerability landscape of Ethereum smart contracts.

Received: 9 December 2023; Accepted: 17 September 2024
Published online: 08 October 2024

References

1. Wood, G. et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper***151**, 1–32 (2014).
2. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Bitcoin*.https://bitcoin.org/bitcoin pdf (2008).
3. Mackintosh, J. Defi is crypto's wall street, without a safety net. *Wall Street Journal* (2021).
4. Alharby, M., Aldweesh, A. & Van Moorsel, A. Blockchain-based smart contracts: A systematic mapping study of academic research (2018). In *2018 International Conference on Cloud Computing, Big Data and Blockchain (ICCBB)*, 1–6 (IEEE, 2018).
5. Dannen, C. *Introducing Ethereum and solidity* Vol. 1 (Springer, 2017).
6. Lyandres, E., Palazzo, B. & Rabetti, D. Initial coin offering (ICO) success and post-ICO performance. *Manage. Sci.***68**, 8658–8679 (2022).
7. Baudier, P., Chang, V. & Arami, M. The impacts of blockchain on innovation management: Sectoral experiments. *J. Innovat. Econ. Manag.***37**, 1–8 (2022).
8. Zamani, E., He, Y. & Phillips, M. On the security risks of the blockchain. *J. Comput. Inf. Syst.***60**, 495–506 (2020).
9. Singh, A., Parizi, R. M., Zhang, Q., Choo, K.-K.R. & Dehghantanha, A. Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. *Comput. Secur.***88**, 101654 (2020).
10. Scharfman, J. Decentralized autonomous organization (dao) fraud, hacks, and controversies. In *The Cryptocurrency and Digital Asset Fraud Casebook, Volume II: DeFi, NFTs, DAOs, Meme Coins, and Other Digital Asset Hacks*, 65–106 (Springer, 2024).
11. He, D. et al. Smart contract vulnerability analysis and security audit. *IEEE Netw.***34**, 276–282 (2020).
12. Sayeed, S., Marco-Gisbert, H. & Caira, T. Smart contract: Attacks and protections. *IEEE Access***8**, 24416–24427 (2020).
13. Zhuang, Y. et al. Smart contract vulnerability detection using graph neural networks. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 3283–3290 (2021).
14. Xing, C. et al. A new scheme of vulnerability analysis in smart contract with machine learning. *Wireless Networks* 1–10 (2020).
15. Zhou, H., Milani Fard, A. & Makanju, A. The state of ethereum smart contracts security: Vulnerabilities, countermeasures, and tool support. *J. Cybersecur. Privacy***2**, 358–378 (2022).
16. Huang, Y., Bian, Y., Li, R., Zhao, J. L. & Shi, P. Smart contract security: A software lifecycle perspective. *IEEE Access***7**, 150184–150202 (2019).
17. Wen, Y., Liu, M., Yang, X., Yang, T. & Chang, V. Bua: A blockchain-based unlinkable authentication scheme for mobile IoT. *Enterprise Inf. Syst.***18**, 2243616 (2024).
18. Aufiero, S. et al. Dapps ecosystems: Mapping the network structure of smart contract interactions. *arXiv preprint*arXiv:2401.01991 (2024).
19. Ibba, G. et al. Mindthedapp: a toolchain for complex network-driven structural analysis of ethereum-based decentralised applications. *IEEE Access* (2024).

20. Zou, W. et al. Smart contract development: Challenges and opportunities. *IEEE Trans. Software Eng.***47**, 2084–2106. https://doi.org/10.1109/TSE.2019.2942301 (2021).
21. Mohanta, B. K., Panda, S. S. & Jena, D. An overview of smart contract and use cases in blockchain technology. In *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1–4 (IEEE, 2018).
22. Blei, D. M., Ng, A. Y. & Jordan, M. I. Latent dirichlet allocation. *J. Mach. Learn. Res.***3**, 993–1022 (2003).
23. Wallach, H. M. Topic modeling: Beyond bag-of-words. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, 977-984, https://doi.org/10.1145/1143844.1143967 (Association for Computing Machinery, New York, NY, USA, 2006).
24. Bakalov, A., McCallum, A., Wallach, H. & Mimno, D. Topic models for taxonomies. In *Proceedings of the 12th ACM/IEEE-CS joint conference on Digital Libraries*, 237–240 (2012).
25. Wang, W. et al. Neural labeled LDA: A topic model for semi-supervised document classification. *Soft. Comput.***25**, 5633–5643. https://doi.org/10.1007/s00500-021-06310-2 (2021).
26. Wei, Y., Wang, W., Wang, B., Bo, Y. & Liu, Y. A method for topic classification of web pages using lda-svm model. In *Proceedings of the 3rd International Conference on Information Science and Control Engineering (ICISCE)*, 318–321, https://doi.org/10.1007/978-981-10-6445-6_64 (Springer, 2017).
27. Liu, L., Tang, L., Dong, W., Yao, S. & Zhou, W. An overview of topic modeling and its current applications in bioinformatics. *Springerplus***5**, 1–22 (2016).
28. Yau, C.-K., Porter, A., Newman, N. & Suominen, A. Clustering scientific documents with topic modeling. *Scientometrics***100**, 767–786 (2014).
29. Asuncion, H. U., Asuncion, A. U. & Taylor, R. N. Software traceability with topic modeling. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, 95–104,https://doi.org/10.1145/1806799.1806817 (2010).
30. Bistarelli, S., Faloci, F. & Mori, P. .chain: automatic coding of smart contracts and user interfaces for supply chains. In *2021 Third International Conference on Blockchain Computing and Applications (BCCA)*, 164–171, https://doi.org/10.1109/BCCA53669.2021.9656987 (2021).
31. Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M. & Lee, H.-N. Systematic review of security vulnerabilities in ethereum blockchain smart contract. *IEEE Access***10**, 6605–6621 (2022).
32. Chang, V. et al. How blockchain can impact financial services-the overview, challenges and recommendations from expert interviewees. *Technol. Forecast. Soc. Chang.***158**, 120166 (2020).
33. Hanif, H., Nasir, M. H. N. M., Ab Razak, M. F., Firdaus, A. & Anuar, N. B. The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. *J. Netw. Comput. Appl.***179**, 103009 (2021).
34. Vacca, A., Fredella, M., Di Sorbo, A., Visaggio, C. A. & Piattini, M. Functional suitability assessment of smart contracts: A survey and first proposal. *J. Softw. Evolut. Process*https://doi.org/10.1002/smr.2636 (2023).
35. Hu, T. et al. Transaction-based classification and detection approach for ethereum smart contract. *Inf. Process. Manage.***58**, 102462. https://doi.org/10.1016/j.ipm.2020.102462 (2021).
36. Tian, G. et al. Smart contract classification with a bi-lstm based approach. *IEEE Access***8**, 43806–43816. https://doi.org/10.1109/ACCESS.2020.2977362 (2020).
37. Shi, C. et al. A bytecode-based approach for smart contract classification. *arXiv preprint*arXiv:2106.15497 (2021).
38. Ferreira Torres, C., Baden, M., Norvill, R. & Jonker, H. Ægis: Smart shielding of smart contracts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2589–2591 (2019).
39. Dingman, W. et al. Classification of smart contract bugs using the nist bugs framework. In *2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA)*, 116–123, https://doi.org/10.1109/SERA.2019.8886793 (2019).
40. Camino, R., Torres, C. F. & State, R. A data science approach for honeypot detection in ethereum. *CoRR***abs/1910.01449** (2019). arXiv:1910.01449.
41. Taghavi, M., Bentahar, J., Otrok, H. & Bakhtiyari, K. A reinforcement learning model for the reliability of blockchain oracles. *Expert Syst. Appl.***214**, 119160. https://doi.org/10.1016/j.eswa.2022.119160 (2023).
42. Geng, Z., Cao, Y., Li, J. & Han, Y. Novel blockchain transaction provenance model with graph attention mechanism. *Expert Syst. Appl.***209**, 118411. https://doi.org/10.1016/j.eswa.2022.118411 (2022).
43. Liu, C. et al. Reguard: Finding reentrancy bugs in smart contracts. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, 65–68 (2018).
44. Jiang, B., Liu, Y. & Chan, W. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 259–269 (IEEE, 2018).
45. Tang, X., Du, Y., Lai, A., Zhang, Z. & Shi, L. Deep learning-based solution for smart contract vulnerabilities detection. *Sci. Rep.***13**, 20106 (2023).
46. Zhang, Q., Wang, Y., Li, J. & Ma, S. Ethploit: From fuzzing to efficient exploit generation against smart contracts. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 116–126 (IEEE, 2020).
47. Zhang, S., Wang, M., Liu, Y., Zhang, Y. & Yu, B. Multi-transaction sequence vulnerability detection for smart contracts based on inter-path data dependency. In *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*, 616–627 (IEEE, 2022).
48. Xue, Y. et al. xfuzz: Machine learning guided cross-contract fuzzing. *IEEE Trans. Dependable Secure Comput.***21**, 515–529 (2022).
49. Bartoletti, M. & Pompianu, L. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *International Conference on Financial Cryptography and Data Security*, 494–509 (Springer, 2017).
50. Vidal, F. R., Ivaki, N. & Laranjeiro, N. Openscv: An open hierarchical taxonomy for smart contract vulnerabilities. *Empir. Softw. Eng.***29**, 101 (2024).
51. Zhang, P., Xiao, F. & Luo, X. A framework and dataset for bugs in ethereum smart contracts. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 139–150 (IEEE, 2020).
52. Pierro, G. A., Tonelli, R. & Marchesi, M. Smart-corpus: an organized repository of ethereum smart contracts source code and metrics. *arXiv preprint*arXiv:2011.01723 (2020).
53. Ortner, M. & Eskandari, S. Smart contract sanctuary.
54. Ali, O. et al. A review of the key challenges of non-fungible tokens. *Technol. Forecast. Soc. Chang.***187**, 122248 (2023).
55. Ibba, G., Pierro, G. A. & Di Francesco, M. 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)*, 34–40, https://doi.org/10.1109/WETSEB52558.2021.00012 (2021).
56. Jagarlamudi, J., Daumé III, H. & Udupa, R. Incorporating lexical priors into topic models. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 204–213 (2012).
57. Hong, L. & Davison, B. D. Empirical study of topic modeling in twitter. In *Proceedings of the First Workshop on Social Media Analytics*, SOMA '10, 80-88, https://doi.org/10.1145/1964858.1964870 (Association for Computing Machinery, New York, NY, USA, 2010).
58. Rameder, H., Di Angelo, M. & Salzer, G. Review of automated vulnerability analysis of smart contracts on ethereum. *Front. Blockchain***5**, 814977 (2022).
59. Ibba, G., Ortu, M. & Tonelli, R. Smart contracts categorization with topic modeling techniques (2021).
60. Torres, C. F., Schütte, J. & State, R. Osiris: Hunting for integer bugs in ethereum smart contracts. In *Proceedings of the 34th annual computer security applications conference*, 664–676 (2018).

Author contributions

M.O. contributed to the methodological part and data analysis G.I. contributed to the data collection and curation M.O. and G.I. and G.D. contributed to writing the main manuscript G.D. and R.T. and C.C. contributed to the review of the manuscript

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to M.O.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.