NOVA IMS Information Management School

MDSAA

Master Degree Program in

Data Science and Advanced Analytics

Enhancing Smart Contract Security: A Machine Learning Framework Using Natural Language Processing and Unsupervised Techniques

Macovei Andrei

Master Thesis

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

Enhancing Smart Contract Security: A Machine Learning Framework Using Natural Language Processing and Unsupervised Techniques

by

Macovei Andrei

Master Thesis presented as partial requirement for obtaining the Master's degree in Data Science and Advanced Analytics, with a specialization in Business Analytics

Supervised by

Ian James Scott, PhD, NOVA Information Management School

June, 2024

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

[Lisbon, July 12,2024]

DEDICATION

I want to dedicate this thesis to my family for allowing me to pursue this master's degree here and to my fiancée, Atenea, who has supported me.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Professor Ian James Scott for trusting me with the chance to write this thesis under his guidance. Working with the professor was a pleasure, and his advice was useful in finishing my master's thesis.

ABSTRACT

This thesis explored smart contract security using natural language processing and unsupervised machine learning techniques. By analyzing reports from Code4Arena and applying k-means and Latent Dirichlet Allocation, I aimed to identify trends in smart contract usage, vulnerabilities, and potential attack methods. My analysis yielded valuable insights for blockchain developers and cybersecurity professionals. The research identified trends in smart contract security threats and the potential of using machine learning for vulnerability detection. I propose a framework to enhance smart contract security based on real-world case studies and data from various blockchain platforms. This framework, informed by the identified trends, can contribute to building more secure blockchain ecosystems. The results indicate that the developed pipeline can efficiently evaluate various smart contracts, uncovering new vulnerabilities and attack types using a severity score.

KEYWORDS

Natural language processing; Blockchain; Smart contracts; Smart contract security

Sustainable Development Goals (SDG):

TABLE OF CONTENTS

Statement of Integrity i
Dedication ii
Acknowledgements iii
Abstract iv
List of Figures vii
List of Tables viii
List of Abbreviations and Acronyms ix
1. Introduction 1
2. Literature review 2
2.1. Blockchain and smart contracts 2
2.1.1. Blockchain and Smart contract elements 2
Transactions 3
Cryptographic Hash Function 3
2.2. Smart Contracts 4
2.2.1. Oracles 5
2.2.2. Security of Smart Contracts 6
2.2.3. Vulnerability Types of a Smart Contract 6
2.2.4. Smart Contracts Attack Types 7
2.2.5. Smart Contracts Protection Techniques 10
2.2.6. Natural language processing 12
2.2.7. NLP-classified smart contract vulnerabilities 13
2.2.8. Smart contract reports analysis 14
2.2.9. Real word projects 15
2.2.10. Summary and gap 15
3. Methodology 17
3.1. Data Collection 17
3.2. Data Preprocessing 17
3.3. Modelling 18
4. Results and discussion 28
4.1. Results 28
4.2. Discussion 36
5. Conclusions and future works 37
5.1. Conclusions 37
5.2. Future Works 38

Bibliographical References 39
Appendix A 46
Appendix B 47
Appendix c 48
Appendix D 49
Appendix E 50

LIST OF FIGURES

Figure 1 - Simplified Blockchain schema (Bosu et al., 2019) 2
Figure 2 – SHA-256 Hash Algorithm (Baivab Kumar Jena, 2024) 3
Figure 3 – How Smart Contracts Work (Hines, 2021) pag 54 4
Figure 4 – How Oracle works (CoinCodeCap, 2023) 5
Figure 5 – A Classification of Exploitation Based on the Fragile Method (Sayeed et al., 2020) pag 3. 7
Figure 6 – Smart Contract Attacks and Protections Overview. (Sayeed et al., 2020) pag 1. 9
Figure 7 - DDoS attack schema (Kumar et al., 2021) 10
Figure 8 - Smart contract security vulnerabilities and testing procedures. (Gupta et al., 2020) pag 9. 11
Figure 9 - Text Processing Steps before modelling (Analytics Vidhya & Chaitanya Krishna Kasaraneni, 2020) 12
Figure 10 - The process of vulnerability analysis (Li et al., 2023) pag 6. 13
Figure 11 - How a Ponzi Schema Works (Michael Henris, 2019) 14
Figure 12 - Empirical research pseudocode 18
Figure 13 - Preprocessing for the second notebook using tokenize and BERT 20
Figure 14 - K-mean algorithm (Leo, 2024) 21
Figure 15 - Euclidian distance 21
Figure 16 - Elbow Method to determine the k number of clusters 22
Figure 17 - Silhouette Analysis for Optimal k 23
Figure 18 – How Latent Dirichlet Allocation works (Naskar, 2019) 24
Figure 19 - Perplexity formula (Hörster et al., 2007). 24
Figure 20 - Pseudocode Evaluating an LDA Model for Text Analysis 25
Figure 21 - Pseudocode shows cluster vulnerabilities and attack types 26
Figure 22 - Severity Rating (National Institute of Standards and Technology, 2024) 27
Figure 23 - Empirical Study Results 28
Figure 24 - Cluster of topics 29
Figure 25 – Number of potential Vulnerabilities per Cluster 30
Figure 26 - Vulnerabilities and attack types from human detection cluster 31
Figure 27 - Vulnerabilities and attack types from machine detection cluster 31
Figure 28 - Vulnerabilities and attack types from unknown detection cluster 32
Figure 29 - LDA with Non-negative Matrix to extract topics from clusters 32
Figure 30 - Top ten most dangerous vulnerabilities in smart contracts 35
Figure 31 - LDA Perplexity evaluation 35

LIST OF TABLES

Table 1- Vulnerabilities and their predicted severity score 33
Table 2- Smart Contract Vulnerabilities are classified using Severity Score(*Smart Contract Weakness Classification (SWC)*, n.d.). 34

LIST OF ABBREVIATIONS AND ACRONYMS

NLP	Natural Language Processing
SHA-256	Secure Hash Algorithm 256-bit
P2PK	Pay to PubKey
P2PH	Pay to Pubkey Hash
DeFi	Decentralized finance
DAO	Decentralized Autonomous Organization
POS	Part-of-speech
TF-IDF	Term Frequency-Inverse Document Frequency
LSTM	Long short-term memory
HTML	HyperText Markup Language
XML	Extensible Markup Language
CSS	Cascading Style Sheets
BERT	Bidirectional Encoder Representations from Transformers
LDA	Latent Dirichlet Allocation
PCA	Principal Component Analysis
CVSS	Common Vulnerability Scoring System
XSS	Cross Site Scripting
OS	Operating system
DoS	Denial of Service
DDoS	Distributed Denial of Service
dApps	Fake Decentralized Applications

AI	Artificial Intelligence
Fake ICOs	Fake Initial Coin Offerings

1. INTRODUCTION

Blockchain technology has the potential to support the field of financial, public, and social services such as land record management, asset management, educational services, energy conservation, citizen registration systems, patient management, taxation systems, security and privacy enhancements of mobile devices and associated services (Khan Sial, 2019). These platforms offer automated decentralized applications called "smart contracts" that describe the terms of contracts and the guidelines for transactions between buyers and the sellers. Smart contracts offer a number of advantages including precision, effectiveness, transparency and confidence, savings, and also safety (Sendner et al., 2023).

While smart contracts have inherent security features such as immutability and self-execution, code flaws and new attack methods could expose them to vulnerabilities, leading to significant financial losses (Chu et al., 2023).

Smart contracts, are key parts of secure blockchain, they are themselves vulnerable due to coding bugs and new attack methods (Chu et al., 2023). These vulnerabilities can lead to financial losses. Even complex blockchain structures cannot fully protect against these issues (Chu et al., 2023). In fact, many attacks exploit simple bugs in the code (Stuart & Lipton, 2018). Secure coding and audits are crucial to minimize these risks.

These theories propose to examine reports of smart contracts from Code4Arena, a platform that identifies these security weaknesses in smart contracts using natural language processing to identify the most recent vulnerabilities and attack types, knowing that smart contracts still have difficulties. This study covers the knowledge gap in the classification of smart contract vulnerabilities using NLP methods. "How can smart contract vulnerabilities be categorized with NLP?" is the research question.

The aim of this research project is to explore how Natural Language Processing (NLP) and Unsupervised Machine Learning can be used to enhance the categorization of vulnerabilities in smart contract complaints obtained from Code4Arena. The study's objective is to develop a more effective approach for identifying vulnerabilities, especially those arising from interactions between human and machine components within smart contracts, using NLP techniques. Ultimately, this contributes to the broader goal of enhancing blockchain security.

2. LITERATURE REVIEW

Smart contracts are revolutionizing many industries as blockchain technology becomes more widely adopted. However, their inherent immutability also exposes vulnerabilities, making them susceptible to exploitation and potential financial losses. This master's thesis aims to enhance our comprehension of the various vulnerabilities and types of attacks that present a risk to smart contracts. The following chapter will offer a concise overview of blockchain technology and smart contracts, covering attack types, security mechanisms, and natural language processing.

2.1. BLOCKCHAIN AND SMART CONTRACTS

A Blockchain can be considered as a data structure that creates a digital record of data and shares the data among a network of separate entities (Laurence, 2017). The blockchain has six elements which include: blocks, transactions, consenus mechanism, cryptographic hash function, decentralized network and smart contracts (Lin, 2023).

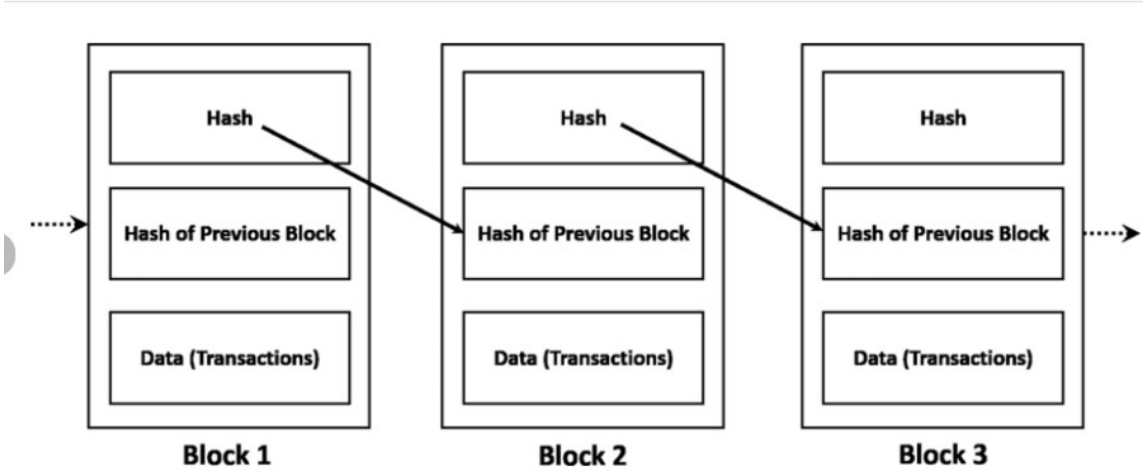


Figure 1 - Simplified Blockchain schema (Bosu et al., 2019)

For all who wish to understand the flow, how to use a blockchain, and most importantly, how to defend a blockchain, understanding the mechanism of how it works is essential. Figure 1 shows how the blockchain works.

2.1.1. Blockchain and Smart contract elements

Blocks

A distributed, chronological database of transactions is called a blockchain. Every node in a peer-to-peer network shares and maintains this database. The way transactions are stored gives rise to the name blockchain. To be more precise, transactions are packaged into blocks that are connected in a chain (Oliva et al., 2020).

The block element is the structure of blockchain that contain a block header and a block body that contain transaction data (Xu et al., 2018). A complete chain of transactions is created because each block contains the ID (identifying code) of the preceding block, allowing each block to locate its prior node (Xu et al., 2018).

Transactions

Transmitting data over a blockchain system's computer network is all that constitutes a blockchain transaction. Blockchain technology records transactions as unchangeable, time-stamped digital blocks by utilizing peer-to-peer (P2P) networks to create a shared, secure ledger. The transaction denotes an agreement between two parties, which may entail the transfer of real or digital assets or the performance of an obligation (Casino et al., 2019).

Cryptographic Hash Function

Cryptology is the science of secure information transmission and storage in a concealed format (Gupta et al., 2019). The message is "encrypted" by the sender, making its content unreadable to a recipient, then "decrypted" by the recipient, restoring its legibility (Coinbase, n.d.). SHA-256 is one of the oldest and most widely utilized hashing algorithms used in blockchains (Eldin et al., 2023).

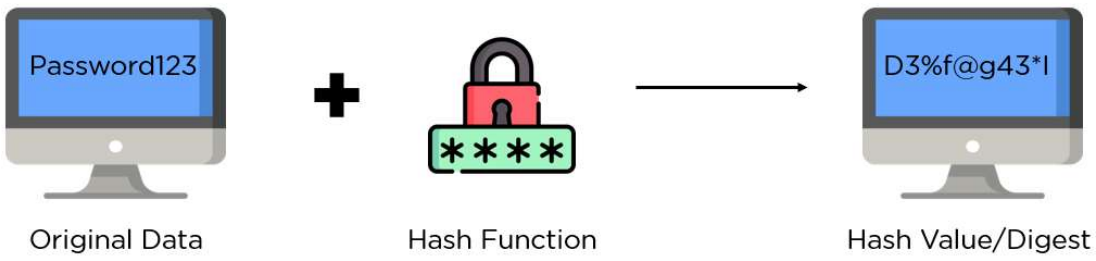


Figure 2 – SHA-256 Hash Algorithm (Baivab Kumar Jena, 2024)

The figure illustrates a simpler paradigm for understanding how hash algorithms work, with the SHA-256 hashing algorithm being one of the most popular in blockchain. Cryptographic hash functions play a crucial role in smart contracts by ensuring data integrity, security, efficiency, and trust in a decentralized environment. These functions are essential for the accurate execution and verification of smart contracts, making them vital for the operation of blockchain technology.

2.2. SMART CONTRACTS

Contracts play a crucial role in the success of a business by outlining legal responsibilities and ownership rights for all parties involved. They establish the guidelines for business transactions, ensure mutual understanding, and settle disputes (Hines, 2021). Blockchain technology incorporates encryption, timestamping, consensus, smart contracts, and other methods to create a decentralized system that safeguards user privacy and data integrity (Wu et al., 2021). Smart contracts are complex programs incorporated into blockchains (Lantz & Cawrey, 2020). The concept of using smart contracts as a digital transaction mechanism to carry out an agreement's terms,which can be observed in figure 3, was initially put forth in the 1990s. The first "smart" contract developed by Bitcoin features P2PK (Pay to Pubkey) and P2PH (Pay to Pubkey Hash) (van Flymen, 2020). The terms of real-world contracts are simply encapsulated and replicated in smart contracts, which are actually containers of code (Taherdoost, 2023).

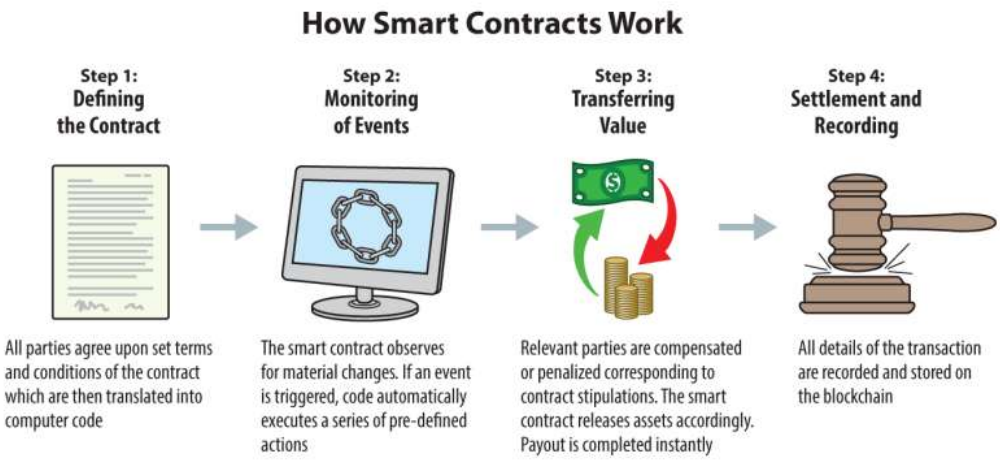


Figure 3 – How Smart Contracts Work (Hines, 2021) pag 54.

Every time a contract method is invoked, the node will add the method-calling transaction to the chain, by figuring out how to execute the transaction and understanding it, the contract state can be updated (Weerapanpisit et al., 2022). After the transaction has been sent over the network, the other nodes will alter their contract states in a similar manner (Weerapanpisit et al., 2022). Smart contracts are incredibly adaptable and have the capacity to manage substantial quantities of data and value while executing immutable logic based on blockchain-deployed code. As a result, a thriving ecosystem of decentralized and trustless apps has emerged, offering several benefits over outdated methods. Additionally, they offer chances for hackers who want to make money by taking advantage of holes in smart contracts (Ethereum.org, 2024).

2.2.1. Oracles

Oracles are intermediary agents that can collect and authenticate data from the outside world and feed it to a blockchain for usage by smart contracts. They could be inbound or external, consensus-based software or hardware. While humans can serve as oracles to start communication between the on-chain and off-chain worlds, doing so is interruptive, time-consuming, and error-prone (Lu et al., 2021). Oracles play a crucial role in smart contracts by providing access to real-world data, ensuring accurate and reliable inputs, and expanding the functionality and applicability of smart contracts across various industries (Beniiche, 2020), as can be seen in figure below. Here's how they function:

- Data Retrieval - Oracles gather information from various sources, including APIs, sensors, and off- chain data providers (Mühlberger et al., 2020).
- Verification- Oracles verify the accuracy and validity of data after retrieving it (Khan et al., 2021).
- On-Chain Transmission - Once confirmed, oracles securely send external data to the blockchain. Smart contracts can then carry out conditional actions depending on this data (trakx, 2024).

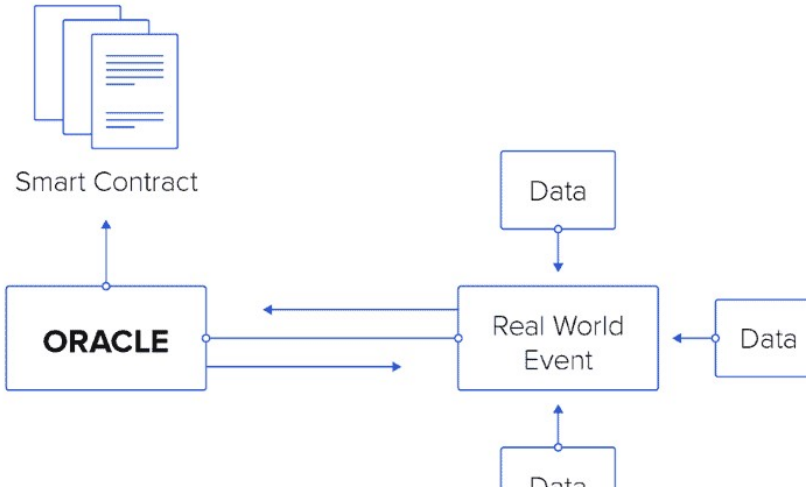


Figure 4 – How Oracle works (CoinCodeCap, 2023)

2.2.2. Security of Smart Contracts

According to (Chaliasos et al., 2023) the necessity for dependable and safe smart contract development has increased due to the rapid growth of the blockchain-based, smart contract-based decentralized finance (DeFi) ecosystem. However, there is an increase in smart contract attacks, which are thought to have resulted in losses of $6.45 billion. Because smart contracts are written in resource-intensive programming languages such as Solidity, they can contain exploitable vulnerabilities or programming errors that are hard to rectify and are frequently missed or found only after the contract has been deployed on the blockchain (Sendner et al., 2023). The immutability of the blockchain has contributed to technology's increased reach in recent years. However, if there is a security issue with the smart contract on the chain after it becomes immutable, clients will suffer significant financial losses. Determining the vulnerabilities in smart contracts is essential. There are two primary categories of smart contract vulnerability detection: automatic detection and detection by humans (Ma et al., 2023). The problem of safeguarding smart contracts is made more difficult by public blockchains like Ethereum.

There are some flaws in the programming languages that the smart contract is built in, including problems with endless loops, arithmetic errors, memory allocation issues, and unintentional mistakes that might result in reentrancy issues and money loss (Porkodi & Kesavaraja, 2023).

Static analysis tools, dynamic analysis tools, or machine learning-based techniques can all be used to analyze a smart contract's vulnerabilities.

- Static Analysis Tools - Tools for static analysis rely on analyzing smart contracts without running them. The static tool's property is that it typically obtains a high coverage rate (Mi et al., 2023).
- Dynamic Analysis Tools - A few tools for dynamic analysis are available. Some rely on constraint solvers' symbolic execution, while others use test cases as program input and examine the outcomes (Mi et al., 2023).
- Machine learning-based Tools - Several studies tried to automate the process of smart contract detection by using machine learning techniques, use simple machine learning algorithms to find vulnerabilities (Mi et al., 2023).

2.2.3. Vulnerability Types of a Smart Contract

As an element of software, smart contracts are inherently fragile, and their vulnerability continues growing (Deng et al., 2023). Hackers can take advantage of a variety of vulnerabilities to cause major issues and financial loss, as shown in the figure below.

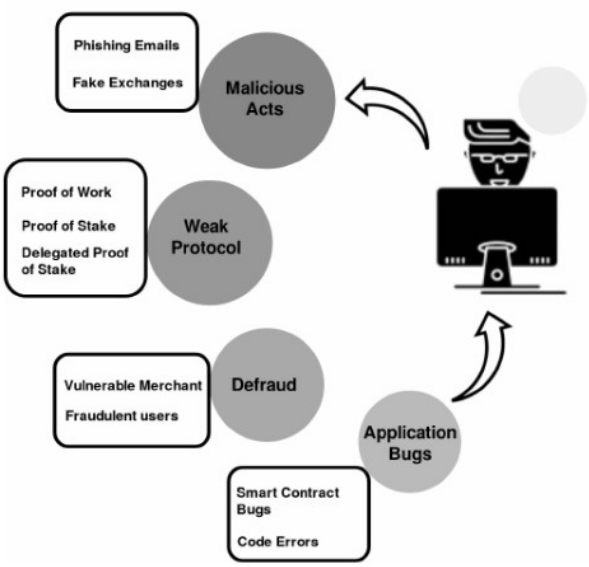


Figure 5 – A Classification of Exploitation Based on the Fragile Method (Sayeed et al., 2020) pag 3.

Vulnerabilities in smart contracts are weaknesses in security or gaps in the contract that can be used to change terms or steal resources. Because of the immutability of blockchain technology, once a smart contract is implemented on the blockchain, it is impossible to alter the code—not even when a vulnerability is found (Chen et al., 2023). Smart contracts are susceptible to a variety of vulnerabilities, with the most common ones listed below being those that hackers can use to compromise a smart contract.

- Malicious Acts - Vulnerable smart contracts can be used by a malicious person to access blockchain data (Gupta et al., 2022).
- Weak Protocol - The blockchain is driven by protocols, as opposed to conventional databases and networks. The flaws in the protocol could result in an abrupt takeover of the system by hackers (Innocent, 2020).
- Defraud - starting a flash loan that led to a smart contract exploit on the DeFi platform, which caused investors and the project's developers to lose money (Federal Bureau of Investigation, 2022).
- Application Bugs - Because of their non-standard software life-cycle, smart contracts make it difficult to upgrade given programs or fix defects by simply releasing new software versions (Destefanis et al., 2018).

2.2.4. Smart Contracts Attack Types

There have been numerous attacks on blockchain and smart contracts in the past few years (Ivanov et al., 2021). A selection of the attacks that can happen with smart contracts is provided below, also a schema.

- Reentrancy attack - One of the frequent risks on the Ethereum blockchain, related to the Solidity programming language, is reentrancy attacks. Attacks occur when an outsider uses a smart contract's external call to force the contract to run extra code by calling itself using a fallback function (Alkhalifah et al., 2021).
- Front-running - Front-running attacks are possible because the users' activities and signatures used to invoke this contract are made public once the relayer submits the transaction (Zhang et al., 2023).
- Integer overflow - In the event of an integer overflow, the software keeps running the next line of code without explicitly throwing an exception. Thus, it is very crucial to test smart contracts for integer overflow errors (Sun et al., 2022).
- Simple logic error - Since formal verification offers a strict mathematical model for smart contract verification to cover the entire program without omitting any situations, it can uncover new or unknown vulnerabilities as well as logical errors (Y. Wang et al., 2021).
- Block gas limit vulnerability - When the gas limit for the smart contract execution is exceeded, an out-of-gas exception is triggered, stopping the current execution (Khan et al., 2021).
- Default visibility - Functions in Solidity can be made external, public, internal, or private via visibility specifiers. When these visibility specifiers are used incorrectly, smart contracts become susceptible to attack and attract attackers (Kushwaha et al., 2022).
- Timestamp dependence (Yatchenko, 2022) **-** The timestamp is often set to the local system time of the miner. The miner can alter this number while still getting the block accepted by other miners, though. When the timestamp is utilized as a trigger condition to carry out particular actions (like delivering money), there is a security risk since the attacker can influence the outcome of the contract by using different block timestamps (S. Wang et al., 2018).
- The DAO, or Decentralized Autonomous Organization, was founded in May 2016 as a venture capital fund focused on the decentralized and cryptocurrency industries. Anyone might deposit Ether to a specific wallet address to receive DAO tokens during the DAO's founding phase (Samreen & Alalfi, 2021).

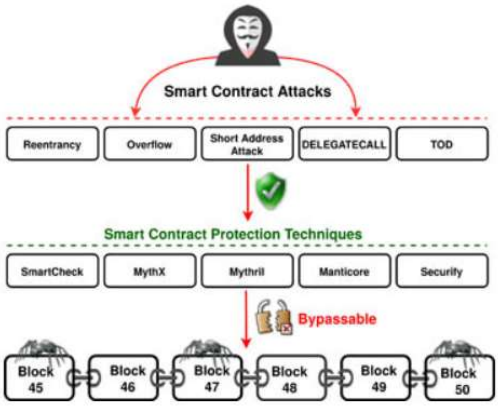


Figure 6 – Smart Contract Attacks and Protections Overview. (Sayeed et al., 2020) pag 1.

An online application vulnerability known as cross-site scripting (XSS) lets an outside entity run a script in the user's browser on the website's behalf. One of the most common flaws on the internet nowadays is cross-site scripting (geeksforgeeks.org, 2019).
A backdoor attack is a method of getting past a computer system's standard security measures and accessing encrypted data.

To enable access to an operating system (OS), data, or application for debugging or other uses, a developer may build a backdoor.
Attackers utilize backdoors installed by software developers as well as backdoors they install on their own as a component of a computer attack (Ben Lutkevich, 2023).

One prevalent risk in smart contracts is race circumstances. When many transactions try to change the same state variable at the same time, a race condition happens. When this occurs, the contract's conclusion becomes unexpected and it is unknown which transaction will be completed first (SolidittySavvy, 2023).

A replay attack vulnerability in many smart contract implementations allows attackers to carry out a replay attack in two similar contracts. This is because many smart contracts use the same method to verify the validity of the signature, but the smart contracts need to properly limit the contents of the signature (baizhenxuan, 2018).
DDOS Attack refers to distributed denial of service assaults, which occur from a variety of places utilizing many systems (geeksforgeeks.org, 2020), as illustrated by the figure below.

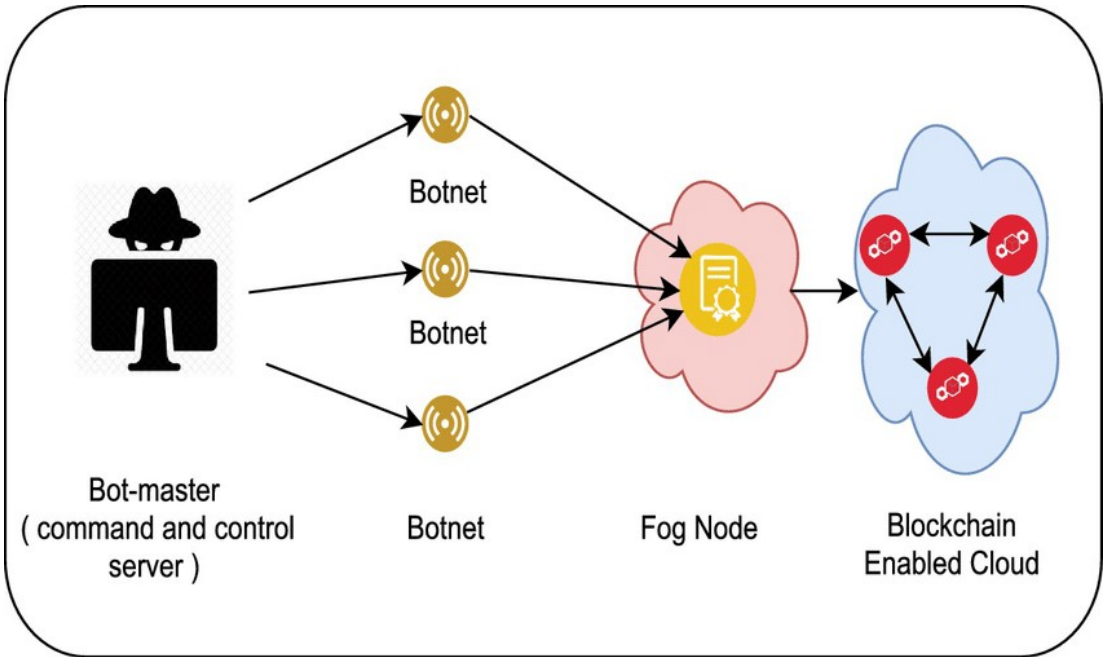


Figure 7 - DDoS attack schema (Kumar et al., 2021)

Phishing schemes deceive people into disclosing personal information, installing malware, and exposing themselves or their companies to criminality (Matthew Kosinski, 2024). Phishing attacks are bogus emails, text messages, phone calls, or websites that are intended to fool consumers into doing the following:

- Downloading Malware
- Sharing sensitive information, like Social Security and credit card numbers, bank account numbers, and login passwords, can put individuals or businesses at risk of cybercrime (Matthew Kosinski, 2024)

2.2.5. Smart Contracts Protection Techniques

In recent years, an immense amount of effort has been put into developing techniques and tools for recognizing vulnerabilities and privacy protection to build and implement safe smart contracts (Y. Wang et al., 2021). Following a brief introduction to smart contract vulnerabilities and forms of smart contract attacks found in sections 2.2.3 and 2.2.4, I go over the primary methods for safeguarding the smart contract. Even if they are unable to identify every vulnerability, therefore nefarious users will continue to search for ways to compromise the system. The fundamental instruments used for protection during column testing are shown in the image below.

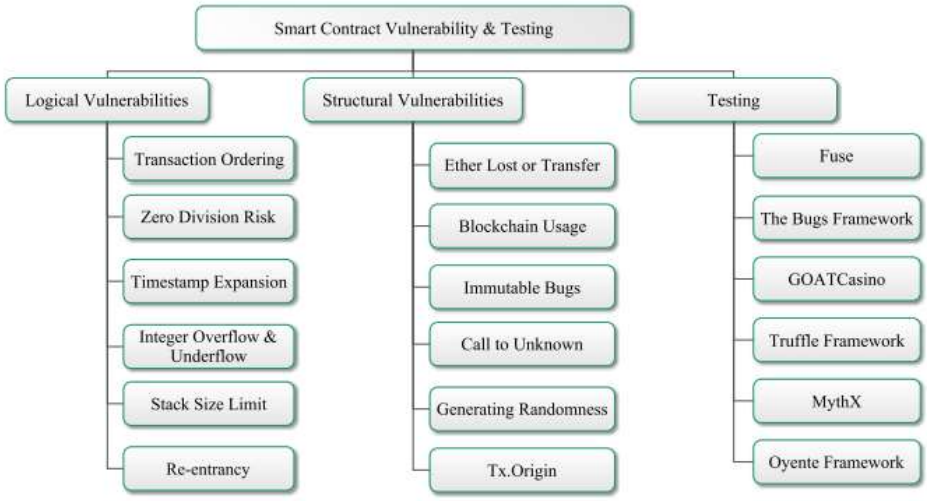


Figure 8 - Smart contract security vulnerabilities and testing procedures. (Gupta et al., 2020) pag 9.

- Fuse - A smart contract created by a developer is often ready for testing but not accessible to the general public. Using a web interface, s/he first sends Fuse an encrypted binary of the smart contract to be tested or the URL of a deployed smart contract (Chan & Jiang, 2018).
- The Bugs Framework - 34,200 Ethereum smart contracts were identified as insecure in an initial review of over one million smart contracts utilizing a single analytical approach for confirming correctness (Ferreira et al., 2020).
- The goal of GOATCasino is to give auditors and developers of smart contracts a sample of the many typical vulnerabilities that are discovered in them so they can see them and not make the same mistakes twice (NCC Group Publication Archive, 2018).
- Truffle Framework - For debugging smart contracts, modern technology is represented by the truffle application framework. It is possible to replay an existing transaction on the blockchain, stepping into and over calls, single-stepping the execution of the bytecode, and other features by providing the unique hash of the transaction to the command truffle debug (Hartel & van Staalduinen, 2019).
- MythX - MythX performs source code and generates bytecode analyses for smart contracts. After that, MythX combines the output from all three tools to enhance the outcomes from various methods. Mythril is one of the components that uses symbolic execution to go through every route conceivable for a smart contract (Kim & Lee, 2020).
- Oyente Framework - is a symbolic execution tool released by Melonport to uncover possible security vulnerabilities such as reentrancy, timestamp dependencies, and logic errors in smart contracts (Shi et al., 2021).

2.2.6. Natural language processing

The field of creating machines that can control written, spoken, and structured human language, or data that mimics human language, is known as natural language processing (NLP). It developed from computational linguistics, which applies computer science to the study of language principles. However, NLP is an engineering field that aims to create technology that can be used to carry out practical tasks, not theoretical frameworks (deeplearning.ai, 2023).

Natural Language Processing (NLP) may be divided into two categories: Natural Language Generation and Natural Language Understanding. NLP advances the process of text generation and comprehension (Khurana et al., 2023). As demonstrated by the figure below.

Important measures to take while processing the text before the modeling phase are as follows:

- Tokenization is the process of locating certain tokens—words and punctuation—in a text (Nadkarni et al., 2011).
- Part-of-speech assignment to individual words, or "POS tagging," is made more difficult in English by homographs (such as "set") and gerunds (verbs with the suffix "ing" that are employed as nouns) (Nadkarni et al., 2011).
- Morphological breakdown of compound words: understanding many medical phrases, such as "nasogastric," requires a breakdown (Nadkarni et al., 2011).
- Lemmatization, or reducing suffixes to reduce a word to a root, is a useful subtask (Nadkarni et al., 2011).

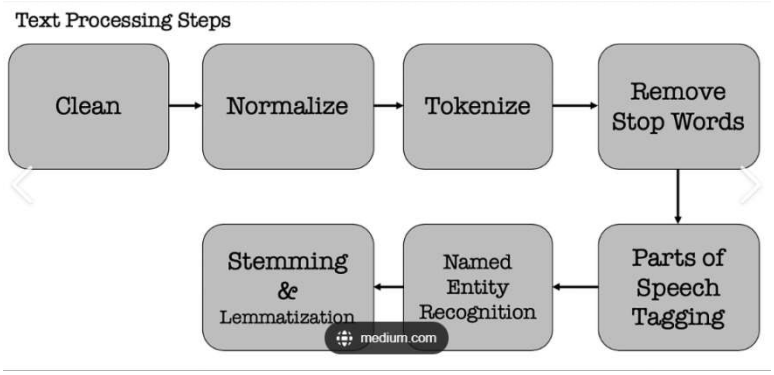


Figure 9 - Text Processing Steps before modelling (Analytics Vidhya & Chaitanya Krishna Kasaraneni, 2020)

The next step after phase cleaning is feature extraction, which may involve:

BERT training involves corrupting text with an arbitrary noising function and learning a model to recover the original text. The conventional Transformer-based neural machine translation architecture, albeit simple, may be considered generalizable (Lewis et al., 2019).

Generalization is one of the main objectives of training NLP models. The conventional approach for evaluation uses train validation-test splits to evaluate the accuracy of the model, including the use of leaderboards to measure progress on a task, because testing "in the wild" is costly and does not allow for quick iterations (Ribeiro et al., 2020).

2.2.7. NLP-classified smart contract vulnerabilities

Smart contract security is becoming more and more of a priority. Still, the studies that have been conducted so far have not come up with a common categorization for smart contract vulnerabilities (Wu et al., 2021). From the viewpoint of technology-driven development, there are two main approaches for detecting smart contract vulnerabilities: traditional techniques, which rely on expert development knowledge, and deep learning methods. Deep learning techniques, however, are data-driven (Deng et al., 2023). Even for known vulnerabilities, the accuracy of some machine learning-based static analysis methods for smart contracts' vulnerability identification is limited. Stated alternatively, the accuracy of rewritten codes reduces dramatically from the original codes (Ashizawa et al., 2021). As apparent in the figure below.

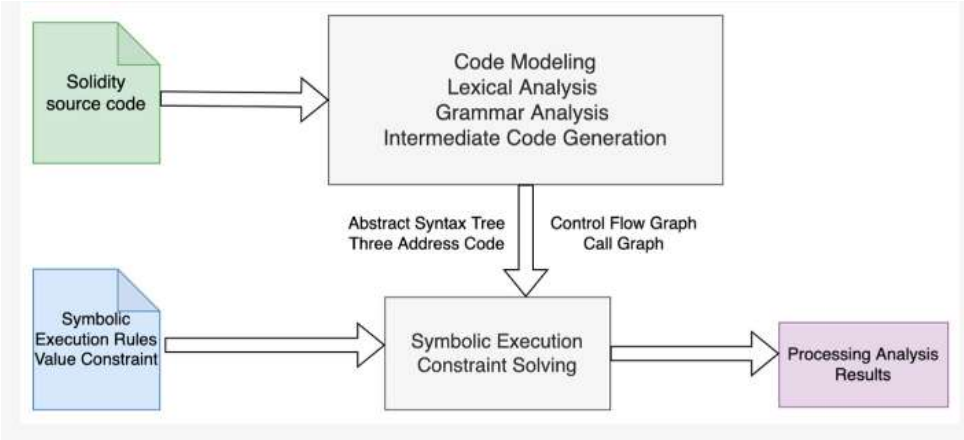


Figure 10 - The process of vulnerability analysis (Li et al., 2023) pag 6.

Natural language models may not have detected several types of attacks and vulnerabilities. Some of these frauds may include:

- Token Sale Scams - This form of smart contract fraud is intended to dupe investors into sending cryptocurrency to scammers' wallets by offering them the chance to buy a new token that will reportedly have a high value in the future (Block Education, 2023a).
- Fake ICOs - Scammers create a smart contract that claims to be associated with an Initial Coin Offering (Block Education, 2023a).
- Fake Decentralized Applications (dApps) - This sort of fraud involves criminals creating a fake dApp that appears to be authentic. They employ marketing strategies to promote the dApp and encourage others to invest in it (Block Education, 2023a).
- Honeypot Scams - A honeypot is a trap designed to detect and divert hostile activities in cybersecurity. For instance, in a "crypto honeypot," the trap is

constructed to particularly identify and monitor would-be attackers seeking to steal or fraudulently gain digital currency (Block Education, 2023c).

- Ponzi Schemes - A Ponzi scheme is a fraudulent financial business in which rewards are given to current investors using cash provided by new investors rather than the operator's profits (Block Education, 2023b) as suggested by the figure below.

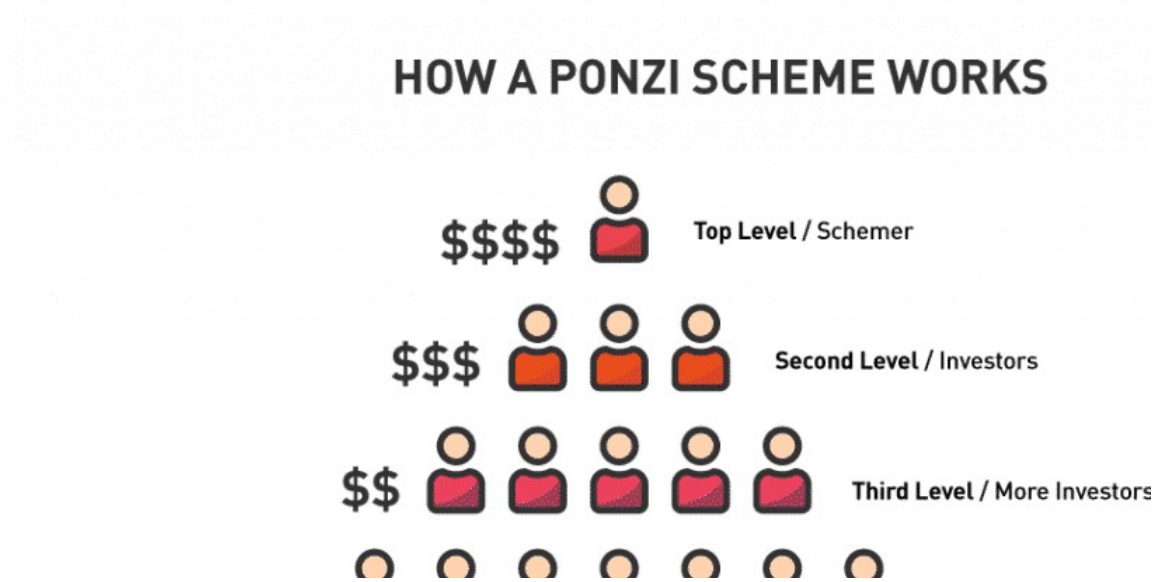


Figure 11 - How a Ponzi Schema Works (Michael Henris, 2019)

2.2.8. Smart contract reports analysis

To find solutions that address security vulnerabilities, poor coding practices, and inefficient code, a smart contract audit entails a thorough examination of the smart contract code for a protocol. Audits contribute to the performance, security, and dependability of Web3 decentralized applications. These actions must be taken in order to compile a report on smart contract vulnerabilities:

- Collect Documentation - including the codebase, whitepaper, architecture, and any other pertinent materials—must be provided to auditors (Chainlink, 2024).
- Automated Testing - Start automated techniques to search code for suspicious or anomalous code that may indicate security flaws (Paul Kirvan, 2023).
- Manual Review - Examine code lines by hand to identify problems that the tools might have overlooked (Paul Kirvan, 2023).
- Classification of Contract Errors - Every error is categorized based on how serious of an exploit it may allow:
  -Crucial: Affects a protocol's ability to operate safely.
  -Major: Logical mistakes and centralization might result in the loss of user cash or protocol control.
  -Medium: Impacts the platform's dependability or performance.

-Minor: Ineffective code that doesn't jeopardize the security of the program (Chainlink, 2024)**.**

- Initial Report - The audit team compiles a final report that includes a list of all problems, vulnerabilities, and tests conducted after comparing the results of automatic and manual examinations (Diana Zozulia, 2023).
- Publish Final Audit Report - All components that were inspected are included in the Checked Items section, and the Findings section offers thorough explanations of the problems found and suggestions for solutions. Furthermore, the Disclaimers make the audit's parameters clear (Parisi, 2023).

2.2.9. Real word projects

This section of the literature study will address smart contract vulnerabilities and real-world application testing.

Vulnerability auditing for smart contracts. Numerous researchers have been putting a lot of effort into this topic and proposing different approaches, as smart contracts are vulnerable. One of them is the Oyente symbolic execution approach, which looks for vulnerabilities in contracts using a number of pre-established patterns (Yang et al., 2020).

Zeus was presented as a smart contract vulnerability program that permitted symbolic modelling and Abstract Interpretation to determine the accuracy of just contracts, and it significantly reduced analysis times. Another framework was presented, which would recognize the contract based on F* and translate the contract bytecode and solid source code into code produced by F*. It does, however, have several drawbacks and does not accommodate numerous poverty-specific grammar elements (Xing et al., 2020).

Another suggestion was to use a deep LSTM model to model a system log as a natural language sequence in order to accomplish the task of anomaly identification in systems.

A system event log is created by a software that adheres to strict logic and control flows and is intended to document system conditions at crucial moments. Analytics is difficult since system logs provide text that is primarily separate and unstructured (Tann et al., 2019).

2.2.10. Summary and gap

Existing research in smart contract security lacks efficient methods for categorizing vulnerabilities. This comment proposes a novel NLP-based approach to address this gap. The key question: can NLP effectively categorize smart contract vulnerabilities? This approach is potentially new and could expedite analysis and improve classification accuracy. It is clear from the literature study that there is ongoing research being done in the field of smart contract security. It is essential to blockchain technology because:

- Oracles are used in secure transactions and smart contract transactions to verify the legitimacy of the contract.

- A crucial feature of smart contracts is their ability to ensure transaction security, even in the event that a malevolent user finds and exploits vulnerabilities.
- One of the main problems with smart contracts is security; once a block is released on the blockchain, it cannot be altered, even if the developer discovers the mistake after deployment and has to wait to release a new block to fix it.
- Even after analyzing the smart contracts, the security process may still be improved by using NLP to analyze the reports and find trends and the primary problems found in them.

The literature study also describes natural language processing technology and explains how to utilize it to find vulnerabilities. Even so, the existing literature has certain limitations and the next steps in my research are as follows:

- My NLP pipeline goes beyond existing tools, discovering previously missed vulnerabilities and providing severity scores to potential attacks and vulnerabilities.
- Determine the vulnerabilities in smart contracts from reports, as not all vulnerabilities are accurately detected by humans.
- Sort the vulnerabilities into those that are recognized by machines and those that are identified by humans.

The smart contract vulnerabilities that are found in reports by employing natural language processing to fill in these gaps will be shown in the next section.

3. METHODOLOGY

This research focuses on utilizing Natural Language Processing (NLP) methods to categorize smart contract vulnerabilities. Although it does not involve a direct qualitative study of human experiences, natural language processing (NLP) offers an effective means to automate the classification of vulnerabilities based on textual data in reports.
Text analysis, which may be either quantitative or qualitative, is used in my research. In qualitative text analysis, texts are closely studied and interpreted to find recurrent ideas, themes, and patterns. Text analysis is helpful in comprehending public conversation, cultural trends, and media messaging (Hassan, 2024).

3.1. DATA COLLECTION

I developed a custom web scraper to gather data for my thesis. It pulled the report text from Code4Arena, and I ended up with 242 reports with raw text that needed to be modelled, to analyze the information the report's content provided properly.

3.2. DATA PREPROCESSING

We have a few phases in the preprocessing phase that are also addressed in detail in the literature review at 2.2.6. In that section, I explain how natural language preprocessing operates and introduce preporcessing processes like lemmatization. Text preprocessing is the term for a set of procedures and methods used on unprocessed text data to get it ready for NLP jobs or further analysis. Text preprocessing is to make the raw text cleaner and more readable, improve the quality of the analyses that come after, and remove extraneous information and noise (Nguyen, 2023). Preprocessing the text essentially means putting it in a format that can be easily analyzed and predicted for the task at hand. Here, a task combines domain and approach. Task: approach + domain (Kavita Ganesan, 2019)

Following the preparatory stage is the modeling phase, which may be approached in several ways depending on the demands. Since my situation involves an unsupervised problem, I have chosen to employ Latent Dirichlet Allocation. Latent Dirichlet Allocation (LDA) automatically uncovers latent topics in large volumes of unlabeled text data, making it suitable for unsupervised situations. My research focuses on unsupervised learning because the text obtained from Code4Arena is unstructured. Unstructured data, sometimes referred to as qualitative data, cannot be handled or evaluated using traditional data tools and procedures. Non-relational databases are best suited for managing unstructured data, since it lacks a predetermined data mode (IBM Cloud Education, 2024).

3.3. MODELLING

The thesis is divided into two parts: one empirical method, and another part that integrates techniques from natural language processing with unsupervised learning. In the data collection section, I explained how I utilize a custom web scraper to extract text from Code4Arena reports and analyze them using NLP.

I conducted research using an empirical approach to identify the smart contracts detected by people and machines. This process differed from the modeling step of the methodology. It is important to briefly examine the information in the reports and consider how it can be used in an empirical method to calculate the number of vulnerabilities discovered in reports identified by people, machines, or unknown sources. I created two lists containing keywords for major detection, as well as a threshold for labeling reports based on the number of vulnerabilities that matched the lists.

```
DEFINE criteria_lists
  human_criteria = List("complex", "subjective", "context-dependent")
  machine_criteria = List("syntax error", "known security issue", "code analysis")

DEFINE empty_keyword_lists
  human_keywords = List()
  machine_keywords = List()

FUNCTION weighted_keyword_score(text, weights):
  score = 0
  FOR keyword IN text:
    IF keyword IN weights:
      score += weights[keyword]
    ELSE:
      score += 1
  RETURN score

DEFINE human_weight(criteria):
  IF criteria IN human_criteria:
    RETURN 2
  ELSE:
    RETURN 1

DEFINE machine_weight(criteria):
  IF criteria IN machine_criteria:
    RETURN 2
  ELSE:
    RETURN 1

FUNCTION label_vulnerabilities(text):
  human_score = weighted_keyword_score(text, human_weights)
  machine_score = weighted_keyword_score(text, machine_weights)
  IF human_score > 2:
    RETURN "Human-Detectable"
  ELSE IF machine_score > 2:
    RETURN "Machine-Detectable"
  ELSE:
    RETURN "Uncertain"
```

Figure 12 - Empirical research pseudocode

I applied Beautiful Soup to extract text from reports, a Python package for extracting data from HTML and XML files (Richardson, L, 2024). However, the websites have JavaScript that detects bots attempting to extract information from the website, which is why I utilized Selenium and time.

I extracted reports content from the data generated during the web scraping stage on Code4Arena website. This stage required concatenating each report's link with the home link. A “for loop”, aided this procedure. A data frame was then generated, which had the report names and their corresponding contents. The preceding section described data cleaning processes. This stage is crucial for future work as it involves collecting and analyzing reports from various websites to identify vulnerabilities, attack types, and patterns in different smart contracts. It is an important part of data collection and modeling.

For the cleaning stage, I load spacy, a free, open-source library for advanced Natural Language Processing (NLP) in Python that can be used to develop information extraction or natural language understanding systems, as well as to pre-process text for deep learning (Explosion, 2024). Also, I utilize various methods to clean the data, such as lowercase text, removing breaks, punctuation, stopwords, digits, as stated in the literature study.

Figure 13 demonstrates how text input is preprocessed with a tokenizer for BERT, making the modeling step much simpler. Tokenization involves separating input text into tokens for computer processing. These tokens are used in natural language processing tasks such as morphological analysis, word class labeling, and parsing (Grefenstette, 1999). BERT pretrains deep bidirectional representations from an unlabeled text by conditioning on the left and right context in all layers (Devlin et al., 2019).

```
FUNCTION tokenize_and_get_embeddings(corpus, tokenizer_name)

  // Initialize variables
  embeddings = EMPTY_LIST

  // Loop through each document in the corpus
  FOR EACH document IN corpus
    tokens = TOKENIZE(document, tokenizer_name)  // Replace TOKENIZE with specific method

    // Apply padding and truncation (optional)
    IF padding AND truncation THEN
      tokens = PAD_and_TRUNCATE(tokens)  // Replace PAD_and_TRUNCATE with specific method
    END IF

    // Get model outputs (assuming a GET_OUTPUTS function for the model)
    outputs = GET_OUTPUTS(tokens)

    // Extract pooled output (replace with specific attribute name if different)
    pooled_output = outputs.pooled_output

    // Convert pooled output to desired format (e.g., list or array)
    document_embedding = CONVERT_TO_FORMAT(pooled_output)

    // Add document embedding to the list
     APPEND document_embedding TO embeddings
  END FOR

  // Combine embeddings into a single data structure (e.g., list or array)
  final_embeddings = COMBINE(embeddings)

  RETURN final_embeddings
```

Figure 13 - Preprocessing for the second notebook using tokenize and BERT

I used K-means clustering and Latent Dirichlet Allocation (LDA) to determine whether the smart contract vulnerabilities were discovered by humans or machines. The next step is to identify the most common vulnerabilities in the reports and prioritize the most serious ones. Because I do not have labeled data, this is an unsupervised problem. I used K-means to group the vulnerabilities based on human, machine, and unknown detection. After clustering, I used LDA to identify the types of vulnerabilities and attacks in each group.

K-Means an unsupervised learning process called clustering seeks to create groups out of the observations in a given dataset. An input of the number of clusters is given. Reducing the total distance of points from their cluster centroids creates the clusters (D, 2020). The k-mean algorithm divides a collection of data into k distinct clusters, using a predetermined k value. The algorithm consists of two distinct phases:

- The first process involves defining k centroids, one for each cluster.
- The next step is to link each point in the data set with the nearest centroid

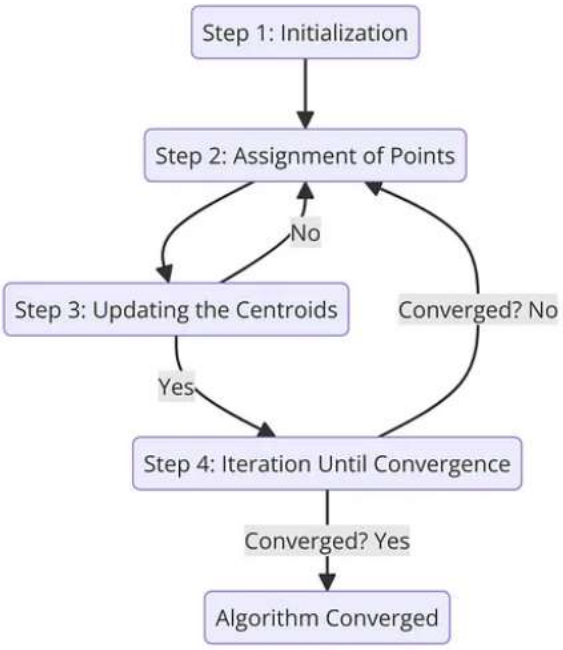


Figure 14 - K-mean algorithm (Leo, 2024)

Euclidean distance is commonly used to determine the distance between data points and centroids (Singh et al., 2013).

$$d(x,\mu) = \sqrt{\sum_{j=1}^{n}(x_j - \mu_{ij})^2}$$

Figure 15 - Euclidian distance

An effective way to figure out how many clusters to employ is the elbow method, which uses visualization to calculate the number of clusters, as it can be seen below. The elbow technique is a visual approach for calculating the number of clusters in a data collection. The "elbow," or point of inflexion on the curve, is a useful indicator of where the underlying model fits best if the line chart resembles an arm (Cui, n.d.).

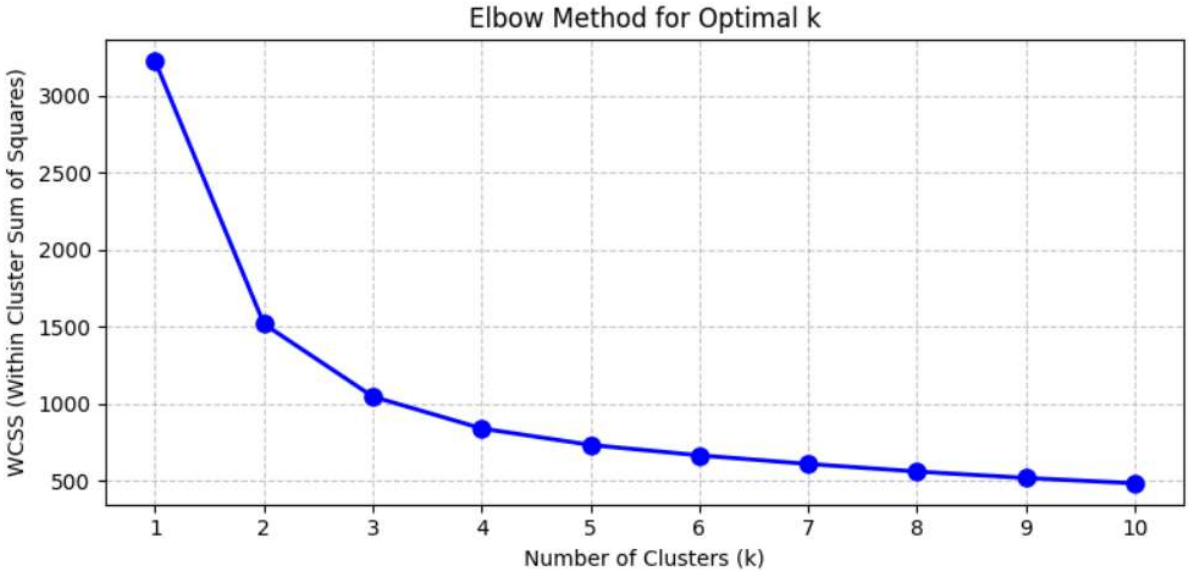


Figure 16 - Elbow Method to determine the k number of clusters

After determining the ideal number of clusters, I picked three since the list of terms for Latent Dirichlet Allocation includes themes that can be determined by both humans and machines, and I chose three for k.

I also conducted silhouette analysis, which can be used to study the distance between the clusters that were created. The silhouette plot illustrates how close each point in one cluster is to points in neighboring clusters, enabling you to visually assess characteristics such as cluster number (Scikit-learn developers, n.d.). The silhouette statistic is a distance-based measurement used in clustering analysis. To calculate the silhouette value at a point, p, in the data set, follow these steps:

- Define AvgDistIn(p) as the average distance between p and the other points in the same cluster. This is the average distance within the cluster that p belongs to.
- Calculate the average distance between p and the points in all the other clusters. If there are k clusters, this will yield k-1 results. Define AvgDistOut(p) as the minimum of these average distances. This represents the minimum of the average distance between clusters.

The silhouette statistic for point p is expressed as s(p) = (AvgDistOut(p) - AvgDistIn(p)) / max(AvgDistOut(p), AvgDistIn(p)). This value falls within the range of -1 to 1, where a higher value indicates that the point is well clustered (Rick Wicklin, 2023).

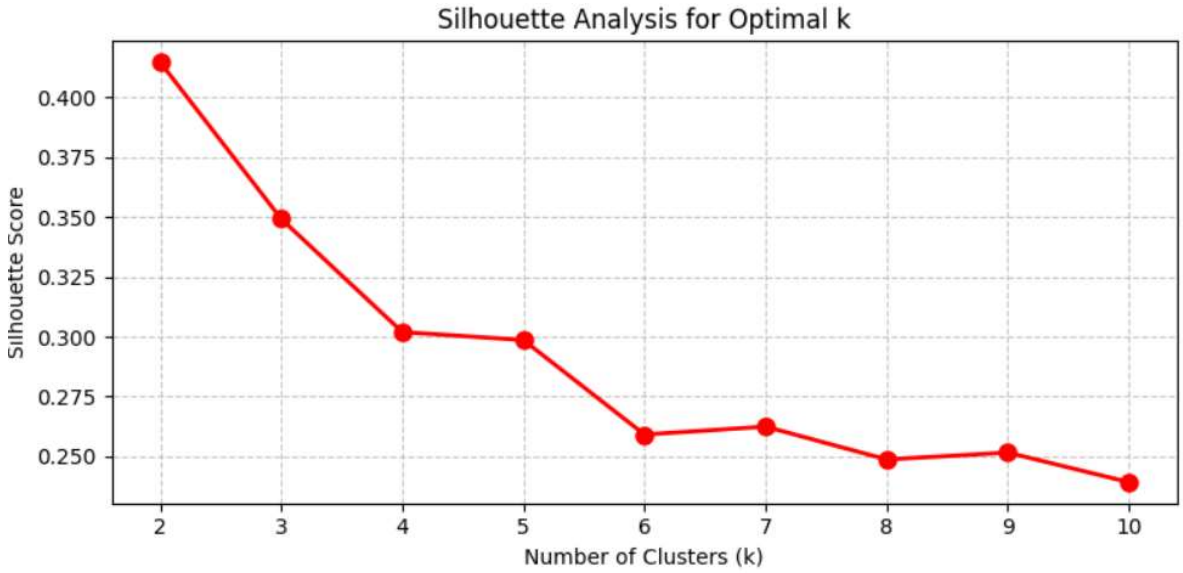


Figure 17 - Silhouette Analysis for Optimal k

After conducting Silhouette analysis, I decided to apply the three groups since three types of detection can be assumed: machine, human, and unknown. For visualization of the clusters, I used PCA. PCA is a statistical approach used to compress and extract meaningful information from multivariate data sets (Deeman Yousif Mahmood, 2018). PCA is widely used in current data analysis, from neuroscience to computer graphics, due to its easy to understand non-parametric approach to extracting useful information from complex datasets (Shlens,2014). For the PCA, I utilized topic embeddings, which are high-dimensional vectors with three groups labelled "Human Detection," "Machine Detection," and "Unknown."

I identified the most common vulnerabilities and attack types found in the reports and generated a top list of the most dangerous vulnerabilities using the severity score, which will be discussed later in the results. To use LDA, I needed to develop the Non-Negative Matrix Factorization, a decomposition technique for high-dimensional data analysis that automatically selects sparse and significant features from a set of nonnegative input vectors. It is also commonly utilized in dimensionality reduction, feature extraction, matrix decomposition, clustering, feature learning, topic recovery, signal enhancement, and other domains, making it a versatile technique (Madala, 2023).

A generative probabilistic model of a corpus is called Latent Dirichlet Allocation (LDA). The fundamental concept is that texts are represented as random mixes across latent subjects, with a word distribution defining each topic (Blei et al., 2003). An unsupervised learning approach called Latent Dirichlet Allocation (LDA) aims to characterize a set of observations as a combination of several categories. The most popular use of LDA is finding a user-specified number of themes that documents in a text corpus share. In this case, every observation is a document, every word's existence represents a feature, and the subjects represent the categories (Amazon Web Services, n.d.).

The steps to implement LDA are as follows:

- Assign each word in a document at random to one of the K pre-defined subjects, where K is the number of topics.
- Regarding every document d:Compute: for each word w in the document.
- P(topic t| document d): The percentage of words attributed to subject t in document d
- P(word w| topic t): The percentage of assignments from words that originate from w in all publications that are related to topic t
- Reassign topic T' to word w, taking into account all other words and their topic assignments, with probability p(t'|d)*p(w|t') (Dutta, 2021).

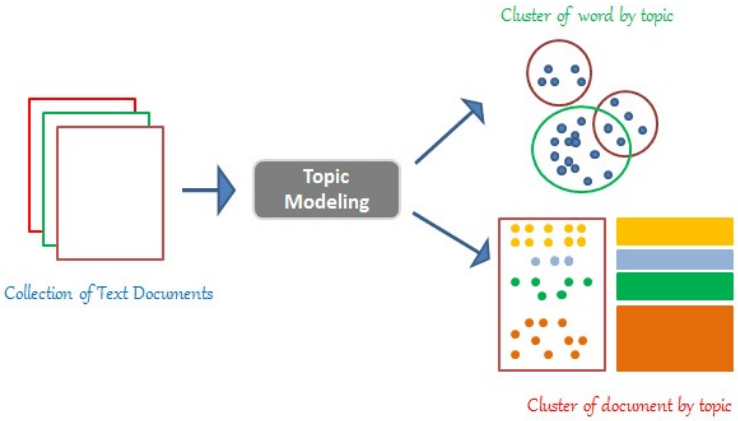


Figure 18 – How Latent Dirichlet Allocation works (Naskar, 2019)

The most common and dangerous vulnerabilities and attack types were discovered for each cluster. This analysis used a higher degree of abstraction in line with official academic norms. I've observed the distribution of potential vulnerabilities across clusters. The Unknown cluster notably contains numerous elements that can be identified by both machines and humans. LDA's performance is commonly evaluated by testing it on secondary tasks such as document categorization or information retrieval.

Another approach is to estimate the likelihood of unobserved held-out documents using the training documents, a measure known as holdout perplexity (Wallach et al., 2009). The perplexity is calculated using the following formula:

$$per(D_{test}) = exp\{-\frac{\sum_{d=1}^{M} \log p(\mathbf{w}_d)}{\sum_{d=1}^{M} N_d}\}$$

Figure 19 - Perplexity formula (Hörster et al., 2007).

M refers to the number of documents in the test sample, while wd represents the words in document d, and Nd represents the total number of words in document d (Stack Exchange Network, 2011). Researchers use hold-out perplexity to evaluate a model performance to

generalize and avoid overfitting training data. It is commonly used in topic modeling and language models to assess how well the model represents the distribution of topics or words (Mimno & McCallum, 2012). This approach can also be used with data that has not been previously analyzed, and also for future reports.

```
**Input:** Preprocessed text corpus (list of documents)

**Output:** Train and test perplexity scores

1. **Split Data (train/test):**
    - Divide the corpus into two sets: training documents (80%) and test documents (20%).
    - Use a random seed (42) for reproducibility.

2. **Feature Extraction (Count Vectorizer):**
    - Create a CountVectorizer object.
        - Filter out features with document frequency (DF) greater than 95% (max_df=0.95).
        - Filter out features with DF less than 2 (min_df=2).
        - Remove common English stop words.
    - Fit the vectorizer on the training documents to learn the vocabulary.
    - Transform both training and test documents into document-term matrices (DTMs) using the fitted vectorizer.

3. **Train LDA Model:**
    - Define the number of topics (e.g., 5).
    - Create a LatentDirichletAllocation object with the desired number of topics and a random seed (42).
    - Train the LDA model on the training DTM.

4. **Evaluate Model Perplexity:**
    - Calculate the perplexity score for the trained LDA model on the training DTM.
    - Calculate the perplexity score for the trained LDA model on the test DTM.
    - Print the train and test perplexity scores.
```

Figure 20 - Pseudocode Evaluating an LDA Model for Text Analysis

As mentioned previously, I automatically extracted the vulnerabilities per cluster. This allowed me to gain a deeper understanding of the smart contract vulnerabilities and the distribution of attack types across clusters, which included Machine, Human, and Unknown detection.

```
1. Import libraries:
   - Import defaultdict from the collections library.

2. Define document vulnerabilities:
   - Create a dictionary called `document_vulnerabilities`.
   - Loop through each document (doc1, doc2, ..., docN).
     - For each document, create a list of vulnerabilities found in that document (including duplicates).
     - Add the document name as a key and the list of vulnerabilities as the value to the `document_vulnerabilities` dictionary.

3. Define topic clusters:
   - Create a list called `topic_clusters`.
   - Loop through each document and assign a cluster label (0, 1, or 2) to the document in the `topic_clusters` list.

4. Define cluster names:
   - Create a dictionary called `cluster_names`.
   - Add mappings between cluster labels (0, 1, 2) and their corresponding names ("Human Detection", "Machine Detection", "Unknown") to the `cluster_names` dictionary.

5. Extract vulnerabilities by cluster:
   - Create a defaultdict called `cluster_vulnerabilities` to store vulnerabilities for each cluster.
   - Loop through each document and its corresponding cluster label from the `topic_clusters` list.
     - Get the list of vulnerabilities for the current document from the `document_vulnerabilities` dictionary.
     - Loop through each vulnerability in the document's vulnerability list.
       - Add the vulnerability to the set of vulnerabilities associated with the current cluster name in the `cluster_vulnerabilities` dictionary.
       - Using a set ensures that duplicate vulnerabilities are not added.

6. Display vulnerabilities by cluster:
   - Loop through each cluster name and its corresponding set of vulnerabilities in the `cluster_vulnerabilities` dictionary.
     - Print the cluster name.
     - Loop through each vulnerability in the current cluster's vulnerability set.
       - Print the vulnerability.
     - Print an empty line to separate clusters.
```

Figure 21 - Pseudocode shows cluster vulnerabilities and attack types

NLP has significantly improved cybersecurity. NLP may be applied in cybersecurity to analyze unstructured text from several perspectives, including lexical, syntactic, and semantic studies (Gutierrez & Namin, 2022). The CVSS provides a qualitative assessment of severity.
CVSS has three metric groups: base, temporal, and environmental. CVSS is an acronym for the Common Vulnerability Scoring System. It is a method for evaluating and ranking disclosed vulnerabilities in a consistent and repeatable manner. CVSS's purpose is to assist in comparing vulnerabilities in various applications and suppliers using a consistent, repeatable, vendor-agnostic technique (Jonathan Risto, 2023). The Base metrics generate a score between 0 and 10, which may subsequently be adjusted by rating the Temporal and Environmental measurements. A CVSS score is also expressed as a vector string, which is a compressed textual representation of the values required to calculate the score (National Institute of Standards and Technology, 2024) which can be seen in the figure below.

CVSS v3.0 Ratings

Severity	Severity Score Range
None*	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Figure 22 - Severity Rating (National Institute of Standards and Technology, 2024)

Research suggests that static analysis technologies can produce false positives and negatives due to their reliance on predefined criteria (Tang et al., 2023). These technologies need more syntax and semantic analysis; their predefined rules can quickly become outdated and inflexible and generalizing to new information (Tang et al., 2023). No matter how inventive blockchain is, even the world's greatest developer is capable of making an inadvertent error that might lead to catastrophic issues (Hacken, 2022).

4. RESULTS AND DISCUSSION

4.1. RESULTS

This study attempts to discover patterns of smart contract vulnerabilities identified in reports, classify them using natural language processing, and then identify the most dangerous vulnerabilities. The classification process was completed using the severity score, which can be reviewed in the previous section, which extensively addressed what it means.

The study responses address the question and provide valuable information, which remains pertinent despite new vulnerabilities or exploitation techniques surfacing.

Also, my analysis takes an empirical approach in which I attempt to determine roughly which smart contract vulnerabilities in reports were identified by humans, machines, or unknown. The empirical research results are shown in the figure below.

```
Label
Machine-Detectable    158
Human-Detectable       82
Uncertain               2
Name: count, dtype: int64
```

Figure 23 - Empirical Study Results

As shown in Figure 23, the majority of the vulnerabilities were found by machines, followed by human detection and an unknown label. Even if I do an empirical investigation first, it is beneficial to understand how previous vulnerabilities were identified. The percentage calculation, which will assist in determining the percentage, is provided below.

Percentage=(Measured Value / Actual Value) * 100

Percentage Machine-Detection = (158/242)*100 = 65.29%

Percentage Human-Detection = (82/242)*100 = 33.88%

Percentage Unknown-Dectection =(2/242)*100=0.83%

The NLP technique produces essentially comparable findings, although it must be noted that it is based on the severity score.

The calculus below illustrates the distribution of vulnerabilities into clusters based on human, machine, or unknown detection. This detection relies on NLP and machine learning techniques.

For Machine Detection: 242*(25.21/100)=60.5

For Human Detection: 242*(48.35/100)=117

For Unknown Detection : 242*(26.45/100)=64

After completing the empirical study, I employed two different approaches: K-means, an unsupervised clustering technique, and Latent Dirichlet Allocation, another unsupervised clustering technique. These methods were explained in the literature review and methodology, and the results can be found on the following pages.

The data indicates that human detection was responsible for the majority of vulnerability identifications, with 117 out of 242 detections (48.35%). Machine detection accounted for 60.5 (25.21%), while unknown detection accounted for the remaining 64 (26.35%). This distribution highlights the effectiveness and reliance on machine-based detection tools, as can be seen in the figure below.

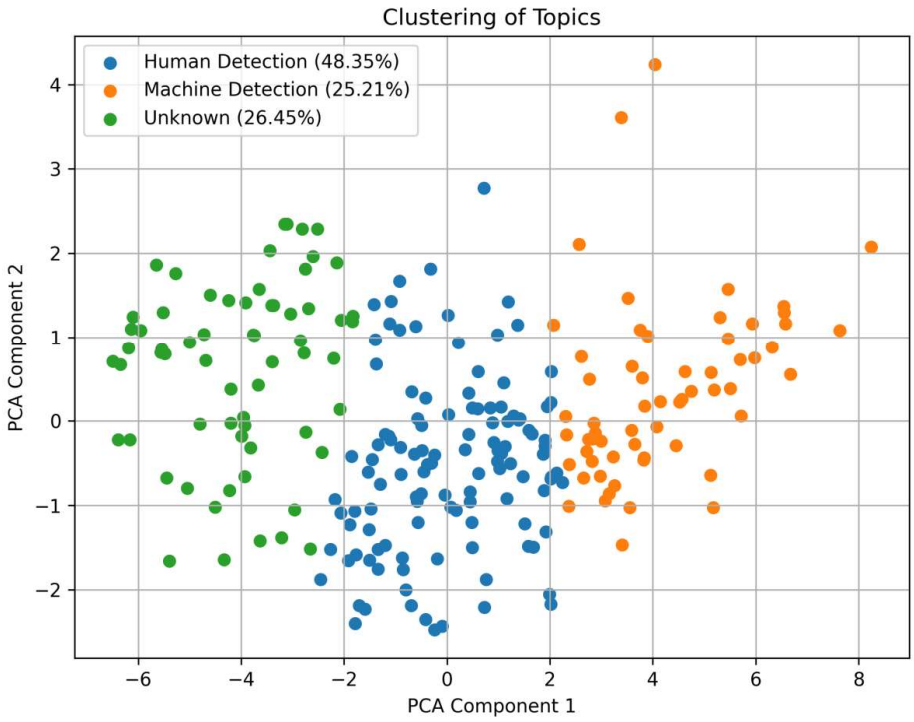


Figure 24 - Cluster of topics

The results indicate that most detection was based on human detection, as can be observed, although the empirical investigation yielded different results. These findings will aid in understanding whether there is an inaccuracy or pattern between smart contract reports and the vulnerabilities identified in them. The following part examines how vulnerabilities in smart contract reports were identified and dispersed by cluster, allowing additional information to be extracted to learn which vulnerabilities are most commonly encountered in reports and how to improve security. Depending on the most prevalent vulnerabilities identified in the reports, we may see a trend and determine the best strategy to improve the security of smart contracts and blockchain.

The study provided valuable insights into the distribution of vulnerabilities across different clusters. The graphic below displays the number of vulnerabilities found per cluster. The first cluster has the highest number of vulnerabilities, even though some of them are human detection. The significant concentration of vulnerabilities in this cluster highlights the need to enhance detection methods and security measures in smart contracts. The number of vulnerabilities detected per cluster can be seen in the image below. It is easy to see that the first cluster contains the most vulnerabilities, even if they are human detection.

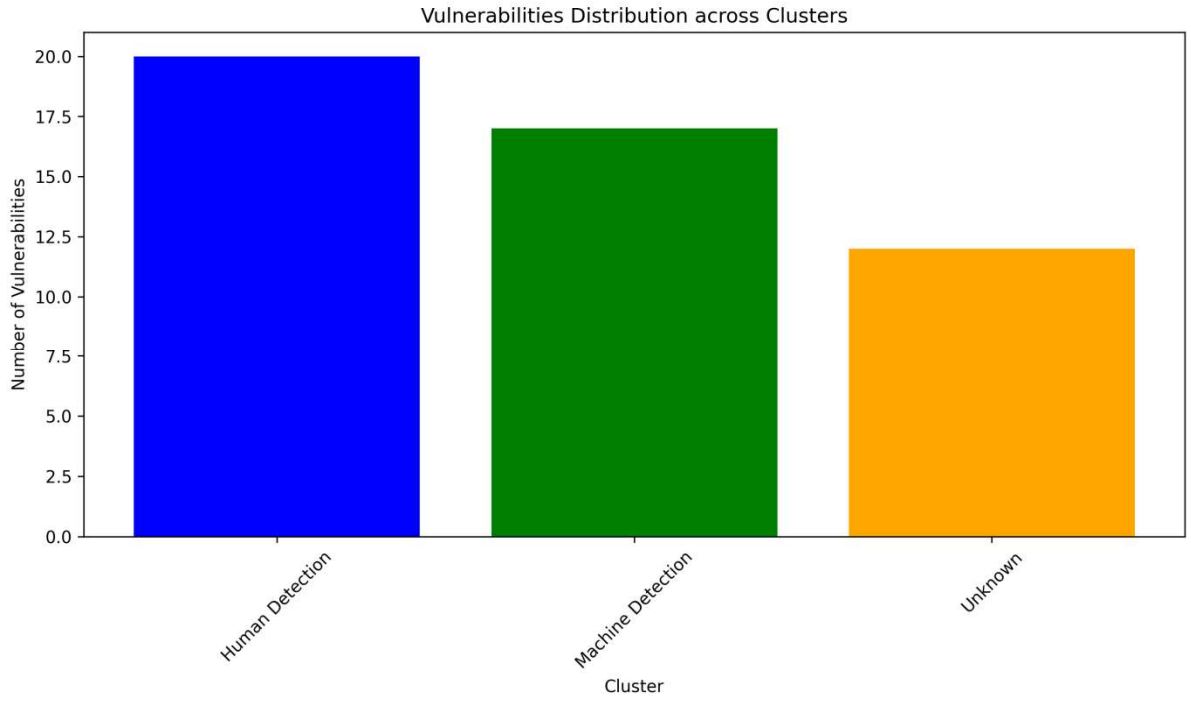


Figure 25 – Number of potential Vulnerabilities per Cluster

Even though the number of vulnerabilities appears to be significant, it is important to note that this visualization includes potential flaws and may show repetitive entries, contributing to the large number of vulnerabilities found in each cluster. The cluster visualization helps in understanding the components of each report, including whether human, machine, or

unknown detection was utilized when writing the smart contract reports. In the second phase, I used LDA to extract vulnerability and attack types for each cluster, as shown in the figures below.

```
Cluster 'Human Detection':
 - insecure deserialization
 - SQL Injection
 - parameter tampering
 - Directory Traversal
 - Open Redirect
 - path traversal
 - csrf
 - session fixation
 - Cross-Site Scripting (XSS)
 - information disclosure
 - xss
 - open redirect
 - Insecure Deserialization
 - Session Fixation
 - Parameter Tampering
 - sql injection
 - Cross-Site Request Forgery (CSRF)
 - cross-site scripting
 - directory traversal
 - Information Disclosure
```

Figure 26 - Vulnerabilities and attack types from human detection cluster

```
Cluster 'Machine Detection':
 - memory corruption
 - privilege escalation
 - authentication bypass
 - Shell Injection
 - Memory Corruption
 - command injection
 - Clickjacking
 - Command Injection
 - buffer overflow
 - Buffer Overflow
 - code injection
 - Click Fraud
 - click fraud
 - Authentication Bypass
 - clickjacking
 - Privilege Escalation
 - shell injection
```

Figure 27 - Vulnerabilities and attack types from machine detection cluster

```
Cluster 'Unknown':
 - Security Misconfiguration
 - Host Header Injection
 - dos
 - Race Condition
 - race condition
 - Denial of Service (DoS)
 - xml external entity
 - XML External Entity
 - host header injection
 - unvalidated input
 - security misconfiguration
 - Unvalidated Input
```

Figure 28 - Vulnerabilities and attack types from unknown detection cluster

In the images above, LDA and Non-negative Matrix helped extract a prioritized list of vulnerabilities and attack types from the clusters.

```
Cluster 0:
Topic 1: file, await, price, memory, token, bytes, position, line, let, public
Topic 2: nft, price, public, recommend, fee, token, returns, protocol, memory, market
Topic 3: memory, public, file, bytes, token, owner, returns, proposal, mitigation, emit
Topic 4: public, memory, vault, bytes, returns, new, file, token, rewards, deposit
Topic 5: submitted, token, watchpug, mitigation, recommended, steps, funds, pool, confirmed, contest

Cluster 1:
Topic 1: submitted, address, uint, owner, commented, watchpug, eth, user, memory, dravee
Topic 2: uint, tokens, address, mitigation, order, memory, token, commented, revert, user
Topic 3: uint, address, revert, vault, nft, token, public, storage, check, commented
Topic 4: uint, tokens, address, return, rewards, await, returns, party, user, external
Topic 5: submitted, tokens, token, watchpug, uint, cmichel, impact, check, pool, user

Cluster 2:
Topic 1: rewards, reward, instances, await, attacker, balance, admin, withdraw, collateral, fee
Topic 2: proposal, vault, fee, eth, buyout, owner, order, price, index, attacker
Topic 3: pool, erc, param, fee, liquidity, missing, price, payable, import, eth
Topic 4: constant, price, onlyowner, await, eth, line, owner, import, valid, pool
Topic 5: instances, index, line, erc, lock, price, asset, votes, owner, pool
```

Figure 29 - LDA with Non-negative Matrix to extract topics from clusters

In the next section, I extracted more potential vulnerabilities, along with already known vulnerabilities and attack types, from report clusters and predict severity scores for each vulnerability or attack type based on Common Vulnerability Scoring System (CVSS).

Vulnerability	Severity Score
remote code execution	10.0
sql injection	10.0
SQL Injection	9.8
Remote Code Execution	9.5
Privilege Escalation	9.1
code injection	9.0
rootkit	9.0
memory corruption	9.0
dll injection	9.0
zero-day exploit	9.0
xss	9.0
cross-site scripting	9.0
xml external entity	9.0
ransomware	9.0
trojan horse	9.0
sqli	9.0
whaling	9.0
Buffer Overflow	8.6
spyware	8.0
cryptojacking	8.0
worm	8.0
botnet	8.0
side-channel attack	8.0
man-in-the-middle attack	8.0
arbitrary file upload	8.0
keylogger	8.0
brute force attack	8.0
logic bomb	8.0
backdoor	8.0
shell injection	8.0
dictionary attack	8.0
spear phishing	8.0
watering hole attack	8.0
social engineering	8.0
identity theft	8.0
impersonation	8.0
mitm attack	8.0
ddos	8.0
distributed denial of service	8.0
layer 7 ddos	8.0
application layer attack	8.0
session hijacking	8.0
iot botnet	8.0
buffer overflow	8.0
xpath injection	8.0
injection attacks	8.0
command injection	8.0
race condition	8.0
broken authentication	8.0
ldap injection	8.0
xml injection	8.0
session fixation	8.0
Insecure Deserialization	8.0
csrf	8.0
file inclusion	8.0
buffer overrun	8.0
format string attack	8.0
Denial of Service (DoS)	7.8
Cross-Site Scripting (XSS)	7.5
path traversal	7.0
data breach	7.0
eavesdropping	7.0
spoofing	7.0
security misconfiguration	7.0
sniffing	7.0
packet sniffing	7.0
directory traversal	7.0
url spoofing	7.0
arp spoofing	7.0
ip spoofing	7.0
mac spoofing	7.0
drive-by download	7.0
pharming	7.0
udp flood	7.0
tcp syn flood	7.0
http flood	7.0
malvertising	7.0
replay attack	7.0
insecure deserialization	7.0
adware	7.0
parameter tampering	7.0
user enumeration	7.0
dos	7.0
smurf attack	7.0
icmp flood	7.0
dns amplification attack	7.0
open redirect	7.0
smishing	7.0
information disclosure	7.0
clickjacking	7.0
unauthorized access	7.0
credential stuffing	7.0
phishing	7.0
host header injection	7.0
vishing	7.0
Cross-Site Request Forgery (CSRF)	6.9
Directory Traversal	6.7
Session Fixation	6.2
privilege escalation	6.0
authentication bypass	6.0
shoulder surfing	6.0
click fraud	6.0
unvalidated input	6.0

Table 1 - Vulnerabilities and their predicted severity score

First table lists the vulnerabilities identified in Code4Arena reports, along with their severity rankings. This table offers a comprehensive overview of the most common types of vulnerabilities and their severity levels. Prioritizing repair efforts based on this information is

crucial for improving the overall security of smart contracts. By evaluating these vulnerabilities and their severity scores, we can identify critical areas that require immediate attention and create targeted mitigation solutions for the most serious threats. This approach not only enhances the security of individual smart contracts, but also contributes to the overall stability of the blockchain ecosystem, thanks to the Smart Contract Weakness.

Classification website, smart contract vulnerabilities can be identified and classified, with severity scores assigned to each one, as shown in the table below (SWC Registry, n.d.). This table helped to predict the severity score for the vulnerabilities identified from smart contract reports.

Vulnerability	**Severity Score**
Reentrancy Attack	High (9/10)
Integer Overflow/Underflow	High (8/10)
Unchecked External Calls	High (8/10)
Front-Running Attacks	Medium (7/10)
Denial of Service (DoS)	Medium (6/10)
Time Manipulation	Medium (6/10)
Lack of Access Control	Medium (6/10)
Gas Limit Vulnerability	Low (4/10)
Transaction-Ordering Dependence (TOD)	Low (3/10)
Timestamp Dependence	Low (3/10)

Table 2 - Smart Contract Vulnerabilities are classified using Severity Score (SWC Registry, n.d.).

As can be seen in the first table, more vulnerabilities were discovered, as well as things that could be considered vulnerabilities and are added to the table with predicted severity score; in the second table, there are only a few of them, but of course, the list of smart contract vulnerabilities can be very long, and the most common were selected and can be considered a reference for the first tabel. These vulnerabilities are considered the most common, but some vulnerabilities are not as common, or a malicious user can try to find a new way to corrupt the smart contract, which is why it is necessary to identify and analyze the reports,

as well as to look for patterns between them and the most dangerous and new vulnerabilities, to secure the new smart contract.

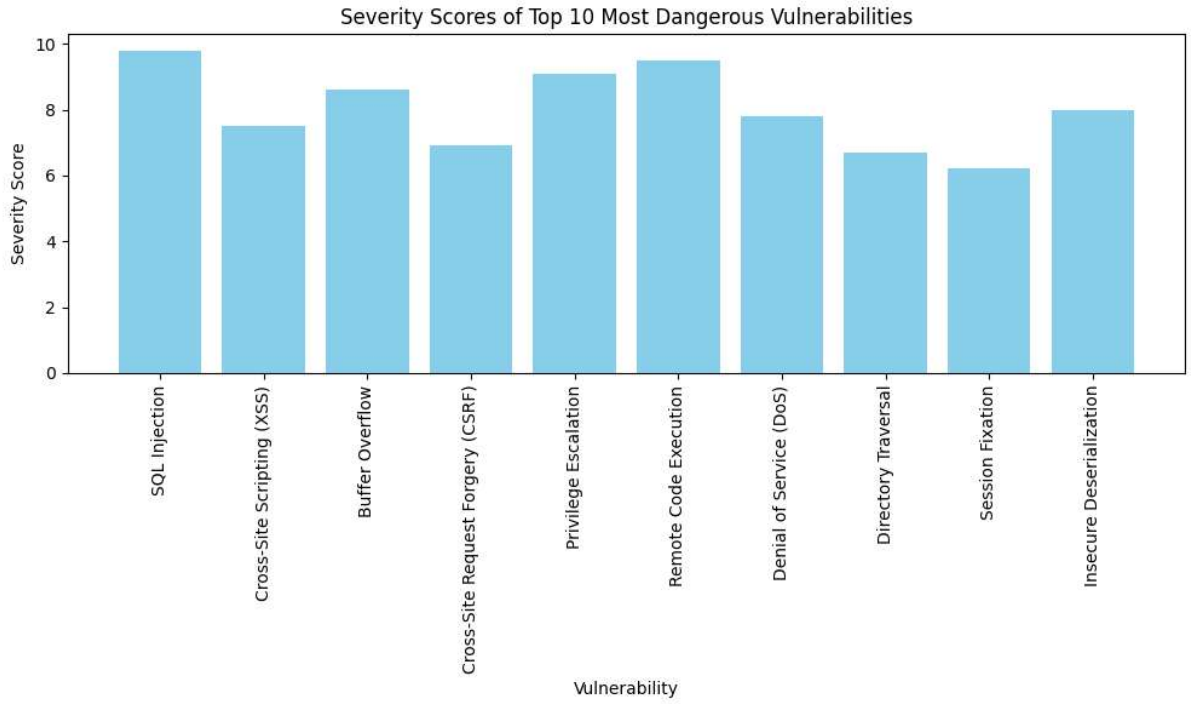


Figure 30 - Top ten most dangerous vulnerabilities in smart contracts

Figure 30 depicts the selection and identification of the most dangerous vulnerabilities discovered in smart contract reports, the severity score was previously explained, and most of them were presented in the literature review as to why they are the most dangerous vulnerabilities. Automatic selection determined that the most dangerous smart contract vulnerabilities and attack types detected in the reports were SQL injection, cross-site scripting, buffer overflow, Cross-Site Request Forgery, privilege escalation, remote code execution, DDOS, directory traversal, session fixation, insecure deserialization. I evaluated the LDA model performance using perplexity and hold-out perplexity. This will help determine how well the model performs on unseen data. The train perplexity (2438.93) is lower than the test perplexity (3674.72), indicating a good fit to training data. However, the increased perplexity on the test set suggests that the model performance suffers with new input, possibly due to overfitting, as can be observed below.

```
Train Perplexity: 2438.925202521219
Test (Hold-Out) Perplexity: 3674.719464666332
```

Figure 31 - LDA Perplexity evaluation

4.2. DISCUSSION

As was discussed in the previous chapter of results, the patterns and vulnerabilities that were identified using unsupervised machine learning techniques combined with natural language processing show important insights of potential vulnerabilities and create a new framework to be used and optimized for further work.

Using natural language processing (NLP) on the reports from smart contracts can help find trends and methods by which attackers can exploit vulnerabilities. Having this information on hand might be beneficial even if the smart contract audit has already been completed.

As shown in the results section, I have developed an NLP pipeline that can effectively identify the most serious vulnerabilities and attack types of smart contracts. This pipeline can be adapted in various ways to pinpoint vulnerabilities or patterns that can aid cybersecurity experts in enhancing smart contract security. It is important to note that once a smart contract is deployed, it cannot be modified until the next smart contract is created, making it critical to ensure its security before its deployment.

The results from this process can help improve the security of future smart contracts. This approach can be applied to various smart contract reports by following the same steps of cleaning, preprocessing, using Kmeans for clustering, and implementing LDA to identify patterns between vulnerabilities and types of attacks in smart contracts. This method can be utilized to categorize smart contract reports, detect vulnerabilities, and predict their severity scores. Furthermore, using LDA, we can extract the top ten vulnerabilities and attack types from smart contract reports in the results area. With this information, cybersecurity professionals and blockchain engineers may be more cautious when testing smart contracts before publishing them to the blockchain, assuring solid protection against hackers. Additionally, it can be used to evaluate the model using perplexity and hold out perplexity.

For further works, as can be observed in the next section, I can add to be developed an NLP pipeline that includes more models, and also to collect data from different websites and not be limited only to Code4Arena. Even if the reports have a good quality, the number of the reports is reduced and my model can have an issue with underfitting the model, which is why I consider that as a discussion; and as for further work that can improve this work is to consider analyzing more smart content.

However, fresh methods to attack and exploit smart contracts will develop, and cybersecurity employees should remain vigilant and conduct investigations even on reports to identify patterns and similar vulnerabilities or kinds of attacks that can be prevented for future smart contracts or future versions of smart contracts.

5. CONCLUSIONS AND FUTURE WORKS

5.1. CONCLUSIONS

Smart contracts are a valuable and simple approach to reaching an agreement in which a transaction may be completed in a few seconds. Even so, flaws and attack types that exploit smart contract vulnerabilities can be identified using smart contract security reports.

Using natural language processing to identify smart contract vulnerabilities and attack types in reports is useful for observing patterns between smart contract vulnerabilities, and cybersecurity employees can be valuable assets in the fight against cybercriminals who attempt to corrupt smart contracts; in this way, cybersecurity employees can assist blockchain developers in creating better and more secure smart contracts.

This thesis offers an alternative method of analyzing smart contracts, which involves studying the language of smart contract reports to find trends, weaknesses, and potential attack vectors that future smart contracts can defend against.

By using K-means clustering and LDA to analyze smart contracts, vulnerabilities and attack types provide a comprehensive perspective of patterns. This method organizes similar vulnerabilities according to severity score and identifies key subjects from textual descriptions. When it does so, it reveals prevalent patterns, severe vulnerabilities, and emerging attack types. Consequently, this strategy allows for improved prioritizing of security operations and money distribution.

Nevertheless, criminals will always find new ways to exploit smart contracts and launch attacks. As such, developers must exercise caution whenever they release a new smart contract, strive to prevent common attack vectors and vulnerabilities, and utilize available tools and strategies to safeguard their creations.

Moreover, unsupervised machine learning models combined with NLP show the capacity to identify potential vulnerabilities in smart contracts. If analyzed, historical data and developing threat trends, these algorithms can predict possible attack types, enabling developers to secure smart contracts to prevent exploitation.

Smart contracts have numerous advantages, but it is crucial for them to be secure. Developers can enhance the reliability of smart contracts by utilizing NLP techniques, machine learning models, and strong cybersecurity protocols. This will ensure their dependability and security in the midst of evolving threats.

5.2. FUTURE WORKS

Smart contracts offer numerous benefits, but it is crucial for them to be secure. Developers can enhance their reliability by utilizing NLP techniques, machine learning models, and robust cybersecurity measures. NLP can improve code clarity and identify ambiguities, while machine learning can detect unforeseen patterns and potential risks in real time. Strong cybersecurity measures such as encryption, multi-factor authentication, and regular audits are essential.

Additionally, formal verification methods can rigorously demonstrate the accuracy of the contract's logic, ensuring consistent behavior. Collaborative research involving blockchain developers, cybersecurity specialists, and data scientists can lead to innovative solutions, while regulatory compliance can safeguard against legal risks. Lastly, it is crucial to train developers and users in standard practices to minimize human error and enhance security.

BIBLIOGRAPHICAL REFERENCES

Alkhalifah, A., Ng, A., Watters, P. A., & Kayes, A. S. M. (2021). A Mechanism to Detect and Prevent Ethereum Blockchain Smart Contract Reentrancy Attacks. *Frontiers in Computer Science*, *3*, 598780. https://doi.org/10.3389/fcomp.2021.598780

Amazon Web Services. (n.d.). *Latent Dirichlet Allocation (LDA) Algorithm—Amazon SageMaker*.Retrieved,March4,2024,from https://docs.aws.amazon.com/sagemaker/latest/dg/lda.html

Analytics Vidhya & Chaitanya Krishna Kasaraneni. (2020). *Understanding NLP Pipeline*. https://medium.com/analytics-vidhya/understanding-nlp-pipeline-9af8cba78a56

Ashizawa, N., Yanai, N., Cruz, J. P., & Okamura, S. (2021). Eth2Vec: Learning Contract-Wide Code Representations for Vulnerability Detection on Ethereum Smart Contracts. *Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, 47–59. https://doi.org/10.1145/3457337.3457841

Baivab Kumar Jena. (2024). *What Is SHA-256 Algorithm: How it Works and Applications | Simplilearn*. Simplilearn.Com. https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm

baizhenxuan. (2018). *Defcon26/docs/Replay Attacks on Ethereum Smart Contracts.md at master · nkbai/defcon26 · GitHub*. GitHub. https://github.com/nkbai/defcon26/tree/master

Ben Lutkevich. (2023). What is a Backdoor Attack? Tips for Detection and Prevention | Definition from TechTarget. Security. https://www.techtarget.com/searchsecurity/definition/back-door

Beniiche, A. (2020). A Study of Blockchain Oracles (arXiv:2004.07140). arXiv. http://arxiv.org/abs/2004.07140

Blei, David. M., Ng, Andrew. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, *3*, pp 993-1022. https://jmlr.csail.mit.edu/papers/v3/blei03a.html

Block Education. (2023a). An Overview of Smart Contracts Scams. *Crypto Adventure*. https://cryptoadventure.com/community/articles/an-overview-of-smart-contracts-scams/

Block Education. (2023b). What Are Crypto Ponzi Schemes? *Crypto Adventure*. https://cryptoadventure.com/community/articles/what-are-crypto-ponzi-schemes/

Block Education. (2023c). What is a HoneyPot Crypto Scam? *Crypto Adventure*. https://cryptoadventure.com/community/articles/short-series-what-is-a-honeypot-crypto-scam/

Bosu, A., Iqbal, A., Shahriyar, R., & Chakraborty, P. (2019). Understanding the motivations, challenges and needs of Blockchain software developers: A survey. *Empirical Software Engineering*, *24*(4), 2636–2673. https://doi.org/10.1007/s10664-019-09708-7

Casino, F., Dasaklis, T. K., & Patsakis, C. (2019). A systematic literature review of blockchain-based applications: Current status, classification and open issues. *Telematics and Informatics*, *36*, 55–81. https://doi.org/10.1016/j.tele.2018.11.006

Chainlink. (2024). *How To Audit a Smart Contract? | Chainlink*. https://chain.link/education-hub/how-to-audit-smart-contract

Chaliasos, S., Charalambous, M. A., Zhou, L., Galanopoulou, R., Gervais, A., Mitropoulos, D., & Livshits, B. (2023). *Smart Contract and DeFi Security: Insights from Tool Evaluations and Practitioner Surveys* (arXiv:2304.02981). arXiv. http://arxiv.org/abs/2304.02981

Chan, W. K., & Jiang, B. (2018). Fuse: An Architecture for Smart Contract Fuzz Testing Service. *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, 707–708. https://doi.org/10.1109/APSEC.2018.00099

Chen, C., Su, J., Chen, J., Wang, Y., Bi, T., Wang, Y., Lin, X., Chen, T., & Zheng, Z. (2023). *When ChatGPT Meets Smart Contract Vulnerability Detection: How Far Are We?* (arXiv:2309.05520). arXiv. http://arxiv.org/abs/2309.05520

Chu, H., Zhang, P., Dong, H., Xiao, Y., Ji, S., & Li, W. (2023). A survey on smart contract vulnerabilities: Data sources, detection and repair. *Information and Software Technology*, *159*, 107221. https://doi.org/10.1016/j.infsof.2023.107221

Coinbase. (n.d.). *Crypto basics—What is cryptography?* Retrieved February 28, 2024, from https://www.coinbase.com/learn/crypto-basics/what-is-cryptography

CoinCodeCap. (2023). *smart contract oracle—CoinCodeCap*. https://coincodecap.com/ethereum-smart-contracts-and-the-oracle-problem

Cui, M. (n.d.). *Introduction to the K-Means Clustering Algorithm Based on the Elbow Method*.

D, V. A. (2020, April 26). K-Means Clustering Algorithm from Scratch. *Machine Learning Plus*. https://www.machinelearningplus.com/predictive-modeling/k-means-clustering/

Deeman Yousif Mahmood. (2018). *Principal Component Analysis (PCA)*. ResearchGate. https://www.researchgate.net/publication/324149818_Principal_Component_Analysis_PCA

deeplearning.ai. (2023). *Natural Language Processing (NLP)—A Complete Guide*. https://www.deeplearning.ai/resources/natural-language-processing/

Deng, W., Wei, H., Huang, T., Cao, C., Peng, Y., & Hu, X. (2023). Smart Contract Vulnerability Detection Based on Deep Learning and Multimodal Decision Fusion. *Sensors*, *23*(16), 7246. https://doi.org/10.3390/s23167246

Destefanis, G., Marchesi, M., Ortu, M., Tonelli, R., Bracciali, A., & Hierons, R. (2018). Smart contracts vulnerabilities: A call for blockchain software engineering? *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 19–25. https://doi.org/10.1109/IWBOSE.2018.8327567

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805). arXiv. http://arxiv.org/abs/1810.04805

Diana Zozulia. (2023). *Smart Contract Audit Process and Why It's Important | 4IRE*. https://4irelabs.com/articles/how-to-audit-smart-contracts/

Dutta, M. (2021). Topic Modelling With LDA -A Hands-on Introduction. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/07/topic-modelling-with-lda-a-hands-on-introduction/

Eldin, S. M. S., El-Latif, A. A. Abd., Chelloug, S. A., Ahmad, M., Eldeeb, A. H., Diab, T. O., Sobky, W. I. A., & Zaky, H. N. (2023). Design and Analysis of New Version of Cryptographic Hash Function Based on Improved Chaotic Maps With Induced DNA Sequences. *IEEE Access*, *11*, 101694–101709. https://doi.org/10.1109/ACCESS.2023.3298545

Ethereum.org. (2024). *Smart contract security*. Ethereum.Org. https://ethereum.org/en/developers/docs/smart-contracts/security/

Explosion. (2024). *spaCy 101: Everything you need to know · spaCy Usage Documentation*. spaCy 101: Everything You Need to Know. https://spacy.io/usage/spacy-101

Federal Bureau of Investigation. (2022). *Cyber Criminals Increasingly Exploit Vulnerabilities in Decentralized Finance Platforms to Obtain Cryptocurrency, Causing Investors to Lose Money*. https://www.ic3.gov/Media/Y2022/PSA220829

Ferreira, J. F., Cruz, P., Durieux, T., & Abreu, R. (2020). *SmartBugs: A Framework to Analyze Solidity Smart Contracts* (arXiv:2007.04771). arXiv. http://arxiv.org/abs/2007.04771

geeksforgeeks.org. (2019). What is Cross Site Scripting (XSS) ? *GeeksforGeeks*. https://www.geeksforgeeks.org/what-is-cross-site-scripting-xss/

geeksforgeeks.org. (2020). *Difference between DOS and DDOS attack*. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-dos-and-ddos-attack/

Grefenstette, G. (1999). Tokenization. In H. Van Halteren (Ed.), *Syntactic Wordclass Tagging* (Vol. 9, pp. 117–133). Springer Netherlands. https://doi.org/10.1007/978-94-015-9273-4_9

Gupta, R., Gupta, P., & Singh, J. (2019). Security and Cryptography. In *Software Engineering for Embedded Systems* (pp. 501–547). Elsevier. https://doi.org/10.1016/B978-0-12-809448-8.00014-X

Gupta, R., Patel, M. M., Shukla, A., & Tanwar, S. (2022). Deep learning-based malicious smart contract detection scheme for internet of things environment. *Computers & Electrical Engineering*, *97*, 107583. https://doi.org/10.1016/j.compeleceng.2021.107583

Gupta, R., Tanwar, S., Al-Turjman, F., Italiya, P., Nauman, A., & Kim, S. W. (2020). Smart Contract Privacy Protection Using AI in Cyber-Physical Systems: Tools, Techniques and Challenges. *IEEE Access*, *8*, 24746–24772. https://doi.org/10.1109/ACCESS.2020.2970576

Gutierrez, L. F., & Namin, A. (2022). ContextMiner: Mining Contextual Features for Conceptualizing Knowledge in Security Texts. *IEEE Access*, *10*, 85891–85904. https://doi.org/10.1109/ACCESS.2022.3198944

Hacken. (2022). *7 Most Common Smart Contract Attacks*. Hacken. https://hacken.io/discover/most-common-smart-contract-attacks/

Hartel, P., & van Staalduinen, M. (2019). *Truffle tests for free—Replaying Ethereum smart contracts for transparency* (arXiv:1907.09208). arXiv. http://arxiv.org/abs/1907.09208

Hassan, M. (2024). Qualitative Research—Methods, Analysis Types and Guide. *Research Method*. https://researchmethod.net/qualitative-research/

Hines, B. (with John Wiley & Sons, Inc). (2021). *Digital finance: Security tokens and unlocking the real potential of blockchain*. John Wiley & Sons, Inc.

Hörster, E., Lienhart, R., & Slaney, M. (2007). Image retrieval on large-scale image databases. *Proceedings of the 6th ACM International Conference on Image and Video Retrieval*, 17–24. https://doi.org/10.1145/1282280.1282283

IBM Cloud Education. (2024). *Structured vs. unstructured data: What's the difference?* https://www.ibm.com/think/topics/structured-vs-unstructured-data

Innocent, O. (2020). *Smart Contract Security: The Attacks and Solutions*. Medium. https://medium.com/xord/smart-contract-security-the-attacks-and-solutions-9d3a48e5604d

Ivanov, N., Lou, J., Chen, T., Li, J., & Yan, Q. (2021). *Targeting the Weakest Link: Social Engineering Attacks in Ethereum Smart Contracts* (arXiv:2105.00132). arXiv. http://arxiv.org/abs/2105.00132

Jonathan Risto. (2023, May 22). *What is CVSS - Common Vulnerability Scoring System*. https://www.sans.org/blog/what-is-cvss/

Kavita Ganesan. (2019, February 23). *All you need to know about text preprocessing for NLP and Machine Learning*. freeCodeCamp.Org. https://www.freecodecamp.org/news/all-you-need-to-know-about-text-preprocessing-for-nlp-and-machine-learning-bc1c5765ff67/

Khan, S. N., Loukil, F., Ghedira-Guegan, C., Benkhelifa, E., & Bani-Hani, A. (2021). Blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-Peer Networking and Applications*, *14*(5), 2901–2925. https://doi.org/10.1007/s12083-021-01127-0

Khan Sial, M. F. (2019). Blockchain Technology – Prospects, Challenges and Opportunities. *IEEE Blockchain Technical Briefs*. https://blockchain.ieee.org/technicalbriefs/june-2019/blockchain-technology-prospects-challenges-and-opportunities

Khurana, D., Koli, A., Khatter, K., & Singh, S. (2023). Natural language processing: State of the art, current trends and challenges. *Multimedia Tools and Applications*, *82*(3), 3713–3744. https://doi.org/10.1007/s11042-022-13428-4

Kim, K. B., & Lee, J. (2020). Automated Generation of Test Cases for Smart Contract Security Analyzers. *IEEE Access*, *8*, 209377–209392. https://doi.org/10.1109/ACCESS.2020.3039990

Kumar, P., Kumar, R., Gupta, G. P., & Tripathi, R. (2021). A Distributed framework for detecting DDoS attacks in smart contract-based Blockchain-IoT Systems by leveraging Fog computing. *Transactions on Emerging Telecommunications Technologies*, *32*(6), e4112. https://doi.org/10.1002/ett.4112

Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., & Lee, H.-N. (2022). Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract. *IEEE Access*, *10*, 6605–6621. https://doi.org/10.1109/ACCESS.2021.3140091

Lantz, L., & Cawrey, D. (2020). *Mastering blockchain: Unlocking the power of cryptocurrencies, smart contracts, and decentralized applications* (First edition). O'Reilly Media, Inc.

Laurence, T. (2017). *Blockchain*. John Wiley & Sons.

Leo, C. (2024, February 18). *The Math Behind K-Means Clustering*. Medium. https://towardsdatascience.com/the-math-and-code-behind-k-means-clustering-795582423666

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension* (arXiv:1910.13461). arXiv. http://arxiv.org/abs/1910.13461

Li, H., Dang, R., Yao, Y., & Wang, H. (2023). A Review of Approaches for Detecting Vulnerabilities in Smart Contracts within Web 3.0 Applications. *Blockchains*, *1*(1), 3–18. https://doi.org/10.3390/blockchains1010002

Lin, P. Y. (2023, March 9). *6 Key Elements of Blockchain Technology*. CFTE. https://blog.cfte.education/6-key-elements-of-blockchain-technology/

Lu, W., Li, X., Xue, F., Zhao, R., Wu, L., & Yeh, A. G. O. (2021). Exploring smart construction objects as blockchain oracles in construction supply chain management. *Automation in Construction*, *129*, 103816. https://doi.org/10.1016/j.autcon.2021.103816

Ma, C., Liu, S., & Xu, G. (2023). HGAT: Smart contract vulnerability detection method based on hierarchical graph attention network. *Journal of Cloud Computing*, *12*(1), 93. https://doi.org/10.1186/s13677-023-00459-x

Madala, S. (2023, March 20). *Non-Negative Matrix Factorization*. Scaler Topics. https://www.scaler.com/topics/nlp/non-negative-matrix-factorization/

Matthew Kosinski. (2024, May 17). *What is a Phishing Attack? | IBM*. https://www.ibm.com/topics/phishing

Mi, F., Zhao, C., Wang, Z., Halim, S. M., Li, X., Wu, Z., Khan, L., & Thuraisingham, B. (2023). *An Automated Vulnerability Detection Framework for Smart Contracts*. https://doi.org/10.48550/ARXIV.2301.08824

Michael Henris. (2019, November 28). *Ponzi schemes: Explanation and samples*. https://www.forexnewsnow.com/top-stories/ponzi-schemes-explanation-and-samples

Mimno, D., & McCallum, A. (2012). *Topic Models Conditioned on Arbitrary Features with Dirichlet-multinomial Regression* (arXiv:1206.3278). arXiv. http://arxiv.org/abs/1206.3278

Mühlberger, R., Bachhofner, S., Castelló Ferrer, E., Di Ciccio, C., Weber, I., Wöhrer, M., & Zdun, U. (2020). Foundational Oracle Patterns: Connecting Blockchain to the Off-Chain World. In A. Asatiani, J. M. García, N. Helander, A. Jiménez-Ramírez, A. Koschmider, J. Mendling, G. Meroni, & H. A. Reijers (Eds.), *Business Process Management: Blockchain and Robotic Process Automation Forum* (pp. 35–51). Springer International Publishing. https://doi.org/10.1007/978-3-030-58779-6_3

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: An introduction. *Journal of the American Medical Informatics Association*, *18*(5), 544–551. https://doi.org/10.1136/amiajnl-2011-000464

Naskar, A. (2019, February 14). *Latent Dirichlet Allocation for Beginners: A high level overview*. ThinkInfi. https://thinkinfi.com/latent-dirichlet-allocation-for-beginners-a-high-level-overview/

National Institute of Standards and Technology. (2024, June 27). *NVD - Vulnerability Metrics*. https://nvd.nist.gov/vuln-metrics/cvss

NCC Group Publication Archive. (2018, March 22). *Discovering Smart Contract Vulnerabilities with GOATCasino*. NCC Group Research Blog. https://research.nccgroup.com/2018/03/22/discovering-smart-contract-vulnerabilities-with-goatcasino/

Nguyen, S. (2023, August 15). Text preprocessing techniques in natural language processing. *StageZero Technologies*. https://stagezero.ai/blog/text-preprocessing-techniques-nlp/

Oliva, G. A., Hassan, A. E., & Jiang, Z. M. (2020). An exploratory study of smart contracts in the Ethereum blockchain platform. *Empirical Software Engineering*, *25*(3), 1864–1904. https://doi.org/10.1007/s10664-019-09796-5

Parisi, O. M., Carlo. (2023, May 22). *How To Audit A Smart Contract: A Deep Dive Into Hacken's Process*. Hacken. https://hacken.io/discover/smart-contract-audit-process/

Paul Kirvan. (2023, May 25). *How to conduct a smart contract audit and why it's needed | TechTarget*. Security. https://www.techtarget.com/searchsecurity/tip/How-to-conduct-a-smart-contract-audit-and-why-its-needed

Porkodi, S., & Kesavaraja, D. (2023). Smart contract: A survey towards extortionate vulnerability detection and security enhancement. *Wireless Networks*. https://doi.org/10.1007/s11276-023-03587-z

Ribeiro, M. T., Wu, T., Guestrin, C., & Singh, S. (2020). *Beyond Accuracy: Behavioral Testing of NLP models with CheckList* (arXiv:2005.04118). arXiv. http://arxiv.org/abs/2005.04118

Richardson, L. (2024). *Beautiful Soup Documentation—Beautiful Soup 4.4.0 documentation*. https://beautiful-soup-4.readthedocs.io/en/latest/

Rick Wicklin. (2023, May 15). *What is the silhouette statistic in cluster analysis?* The DO Loop. https://blogs.sas.com/content/iml/2023/05/15/silhouette-statistic-cluster.html

Samreen, N. F., & Alalfi, M. H. (2021). *A Survey of Security Vulnerabilities in Ethereum Smart Contracts* (arXiv:2105.06974). arXiv. http://arxiv.org/abs/2105.06974

Sayeed, S., Marco-Gisbert, H., & Caira, T. (2020). Smart Contract: Attacks and Protections. *IEEE Access*, *8*, 24416–24427. https://doi.org/10.1109/ACCESS.2020.2970495

Scikit-learn developers. (n.d.). *Selecting the number of clusters with silhouette analysis on KMeans clustering*. Scikit-Learn. Retrieved June 29, 2024, from https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

Sendner, C., Chen, H., Fereidooni, H., Petzi, L., König, J., Stang, J., Dmitrienko, A., Sadeghi, A.-R., & Koushanfar, F. (2023). Smarter Contracts: Detecting Vulnerabilities in Smart Contracts with Deep Transfer Learning. *Proceedings 2023 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium, San Diego, CA, USA. https://doi.org/10.14722/ndss.2023.23263

Shi, C., Xiang, Y., Doss, R. R. M., Yu, J., Sood, K., & Gao, L. (2021). *A Bytecode-based Approach for Smart Contract Classification* (arXiv:2106.15497). arXiv. http://arxiv.org/abs/2106.15497

Shlens, J. (2014). *A Tutorial on Principal Component Analysis* (arXiv:1404.1100). arXiv. http://arxiv.org/abs/1404.1100

Singh, A., Yadav, A., & Rana, A. (2013). K-means with Three different Distance Metrics. *International Journal of Computer Applications*, *67*(10), 13–17. https://doi.org/10.5120/11430-6785

SolidparticleSavvy. (2023, March 31). Race Conditions and Front Running Vulnerabilities in Smart Contracts: A Comprehensive Overview. *Coinmonks*. https://medium.com/coinmonks/race-conditions-and-front-running-vulnerabilities-in-smart-contracts-a-comprehensive-overview-5ea67cfd9d01

Stack Exchange Network. (2011, November 10). *text mining—How to calculate perplexity of a holdout with Latent Dirichlet Allocation?* Cross Validated. https://stats.stackexchange.com/questions/18167/how-to-calculate-perplexity-of-a-holdout-with-latent-dirichlet-allocation

Stuart, D. L., & Lipton, A. B. (2018). *An Introduction to Smart Contracts and Their Potential and Inherent Limitations*. https://corpgov.law.harvard.edu/2018/05/26/an-introduction-to-smart-contracts-and-their-potential-and-inherent-limitations/

Sun, J., Huang, S., Zheng, C., Wang, T., & Zong, C. (2022). Mutation Testing for Integer Overflow in Ethereum Smart Contracts. *TSINGHUA SCIENCE AND TECHNOLOGY*, *27*, 27–40. https://doi.org/1 0. 2 6 5 9 9/T ST. 2 0 2 0. 9 0 1 0 0 3 6

SWC Registry. (n.d.). *Smart Contract Weakness Classification (SWC)*. Retrieved April 9, 2024, from https://swcregistry.io/

Taherdoost, H. (2023). Smart Contracts in Blockchain Technology: A Critical Review. *Information*, *14*(2), 117. https://doi.org/10.3390/info14020117

Tang, X., Du, Y., Lai, A., Zhang, Z., & Shi, L. (2023). Deep learning-based solution for smart contract vulnerabilities detection. *Scientific Reports*, *13*(1), 20106. https://doi.org/10.1038/s41598-023-47219-0

Tann, W. J.-W., Han, X. J., Gupta, S. S., & Ong, Y.-S. (2019). *Towards Safer Smart Contracts: A Sequence Learning Approach to Detecting Security Threats* (arXiv:1811.06632). arXiv. http://arxiv.org/abs/1811.06632

trakx. (2024, June 13). *What Are Blockchain Oracles And Why Are They So Important?* https://trakx.io/resources/insights/what-are-blockchain-oracles/

van Flymen, D. (2020). *Learn blockchain by building one: A concise path to understanding cryptocurrencies*. Apress.

Wallach, H. M., Murray, I., Salakhutdinov, R., & Mimno, D. (2009). Evaluation methods for topic models. *Proceedings of the 26th Annual International Conference on Machine Learning*, 1105–1112. https://doi.org/10.1145/1553374.1553515

Wang, S., Yuan, Y., Wang, X., Li, J., Qin, R., & Wang, F.-Y. (2018). An Overview of Smart Contract: Architecture, Applications, and Future Trends. *2018 IEEE Intelligent Vehicles Symposium (IV)*, 108–113. https://doi.org/10.1109/IVS.2018.8500488

Wang, Y., He, J., Zhu, N., Yi, Y., Zhang, Q., Song, H., & Xue, R. (2021). Security enhancement technologies for smart contracts in the blockchain: A survey. *Transactions on Emerging Telecommunications Technologies*, *32*(12), e4341. https://doi.org/10.1002/ett.4341

Weerapanpisit, P., Trilles, S., Huerta, J., & Painho, M. (2022). A Decentralized Location-Based Reputation Management System in the IoT Using Blockchain. *IEEE Internet of Things Journal*, *9*(16), 15100–15115. https://doi.org/10.1109/JIOT.2022.3147478

Wu, H., Zhang, Z., Wang, S., Lei, Y., Lin, B., Qin, Y., Zhang, H., & Mao, X. (2021). Peculiar: Smart Contract Vulnerability Detection Based on Crucial Data Flow Graph and Pre-training Techniques. *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, 378–389. https://doi.org/10.1109/ISSRE52982.2021.00047

Xing, C., Chen, Z., Chen, L., Guo, X., Zheng, Z., & Li, J. (2020). A new scheme of vulnerability analysis in smart contract with machine learning. *Wireless Networks*. https://doi.org/10.1007/s11276-020-02379-z

Xu, M., Tian, Y., Li, J., Kui, H., & Liu, J. (2018). *Blockchain: An illustrated guidebook to understanding Blockchain*. Skyhorse Publishing.

Yang, Z., Keung, J., Zhang, M., Xiao, Y., Huang, Y., & Hui, T. (2020). Smart Contracts Vulnerability Auditing with Multi-semantics. *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, 892–901. https://doi.org/10.1109/COMPSAC48688.2020.0-153

Yatchenko, D. (2022, December 16). *7 Smart Contract Vulnerabilities & How to Prevent Them [2024]*. PixelPlex. https://pixelplex.io/blog/smart-contract-vulnerabilities/

Zhang, W., Wei, L., Cheung, S.-C., Liu, Y., Li, S., Liu, L., & Lyu, M. R. (2023). Combatting Front-Running in Smart Contracts: Attack Mining, Benchmark Construction and Vulnerability Detector Evaluation. *IEEE Transactions on Software Engineering*, 1–17. https://doi.org/10.1109/TSE.2023.3270117

APPENDIX A

Empirical study

```
# Refine criteria for human-detectable vulnerabilities
human_criteria = ['complex', 'nuanced', 'interpretation', 'subjective', 'context-dependent', 'difficult', 'experience', 'challen

# Refine criteria for machine-detectable vulnerabilities
machine_criteria = ['syntax', 'security', 'automated', 'static', 'analysis', 'code', 'known', 'automatically', 'logical', 'algor

# Expand the lists of keywords
human_keywords = ['complex', 'nuanced', 'interpretation', 'subjective', 'context-dependent', 'difficult', 'experience', 'challen
machine_keywords = ['syntax', 'security', 'automated', 'static', 'analysis', 'code', 'known', 'automatically', 'logical', 'algor

# Weighted scoring
def weighted_keyword_score(text, keyword_weights):
    return sum(keyword_weights.get(word, 1) * (word in text) for word in keyword_weights)

# Stricter weights for human-detectable vulnerabilities
human_weights = {'complex': 2, 'nuanced': 2, 'interpretation': 2, 'subjective': 2, 'context-dependent': 2, 'difficult': 2, 'expe

# Stricter weights for machine-detectable vulnerabilities
machine_weights = {'syntax': 2, 'security': 2, 'automated': 2, 'static': 2, 'analysis': 2, 'code': 2, 'known': 2, 'automatically

# Function to label smart contract reports based on refined criteria and stricter weights
def label_vulnerabilities(text):
    human_score = weighted_keyword_score(text, human_weights)
    machine_score = weighted_keyword_score(text, machine_weights)

    # Stricter threshold for labeling
    threshold = 2.0

    if human_score > threshold:
        return 'Human-Detectable'
    elif machine_score > threshold:
        return 'Machine-Detectable'
    else:
        return 'Uncertain'

# Apply the labeling function to create a new 'Label' column
data['Label'] = data['Report_Content'].apply(label_vulnerabilities)

# Display counts of each label
print(data['Label'].value_counts())
```

APPENDIX B

Launch the web scraper

```
from selenium import webdriver
from bs4 import BeautifulSoup
import time

url = 'https://code4rena.com/reports'

# Initialize the Chrome WebDriver
driver = webdriver.Chrome()
driver.get(url)

# Allow time for dynamic content to load
time.sleep(5)

# Get the page source after content loads
page_source = driver.page_source

# Close the Selenium-controlled browser
driver.quit()

# Parse the loaded content using BeautifulSoup
soup = BeautifulSoup(page_source, 'html.parser')

# Find the desired elements
reports = soup.find_all('div', class_='report-tile')

# Process the reports as needed
print(reports)
```

APPENDIX C

Github Repository link

https://github.com/AndreiM13/Smart-Contracts-Security

Code4Arena Reports List

https://code4rena.com/reports

Code4Arena Report Content Example

https://code4rena.com/reports/2024-08-wildcat

Data frame of raw data extracted from Code4Arena reports

Out[2]:

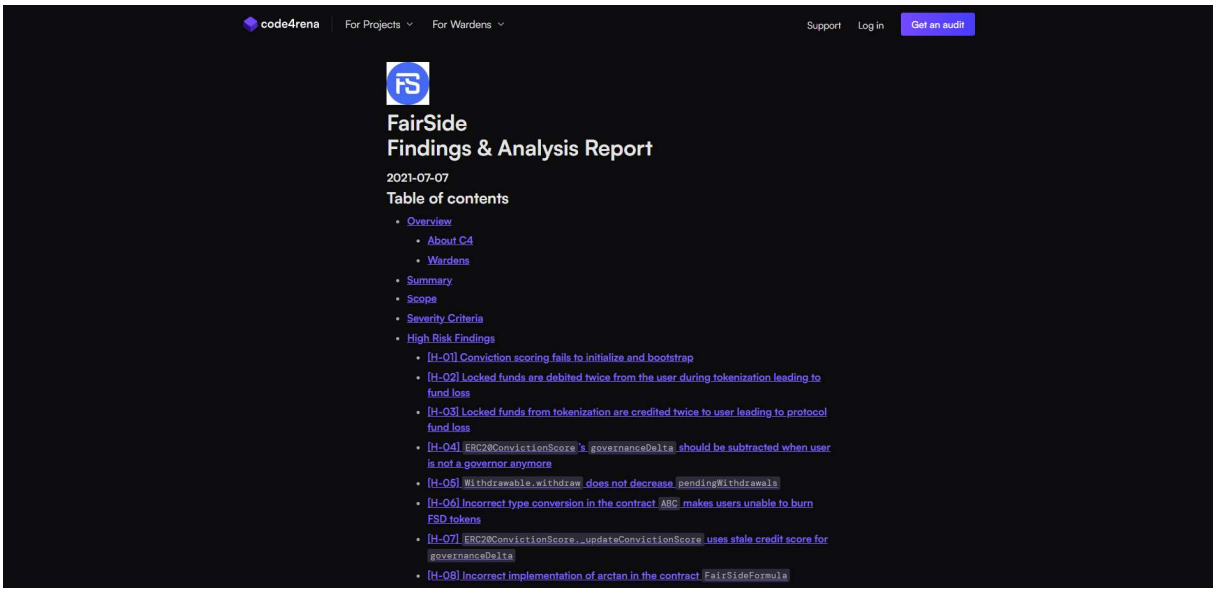
	Report_Title	Report_Content
0	Arcade.xyz →	Arcade.xyz Findings & Analysis Report2023-10-...
1	Livepeer →	Livepeer Onchain Treasury Upgrade Findings & ...
2	Ondo Finance →	Ondo Finance Findings & Analysis Report2023-1...
3	Centrifuge →	Centrifuge Findings & Analysis Report2023-10-...
4	Canto →	veRWA Findings & Analysis Report2023-10-11Tab...
...	...	...
237	Marginswap →	Marginswap Findings & Analysis Report2021-05-...
238	ElasticDAO →	ElasticDAO Findings & Analysis Report2021-03-...
239	Slingshot Finance →	Slingshot Finance Findings & Analysis Report2...
240	Nested Finance →	Nested Finance contest Findings & Analysis Re...
241	BadgerDAO →	BadgerDAO Zaps contest Findings & Analysis Re...

242 rows × 2 columns

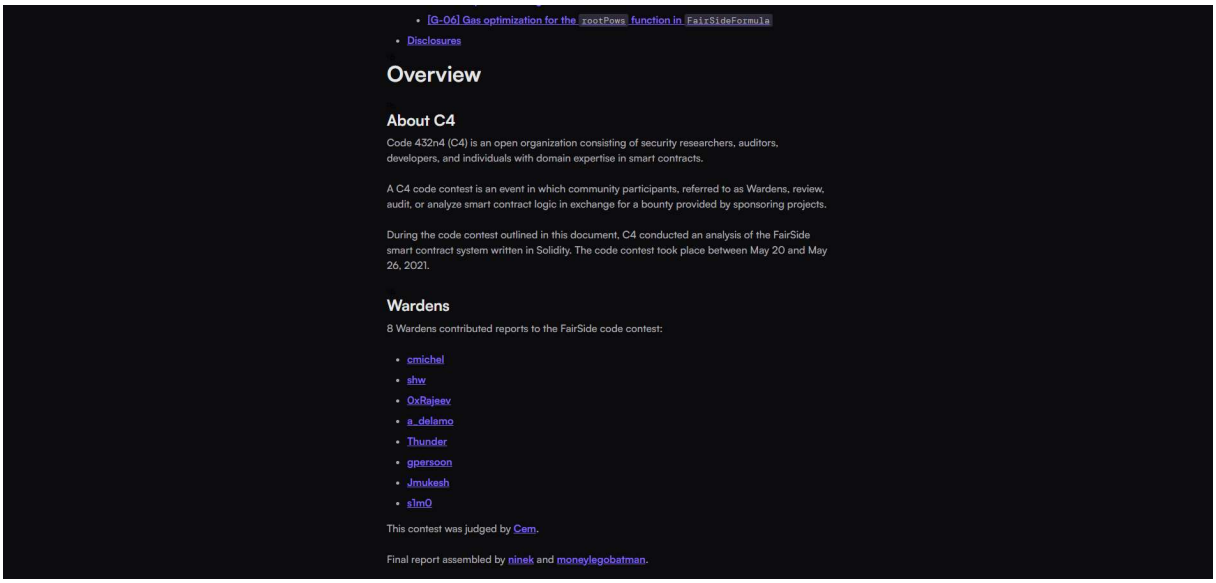
APPENDIX D

Code4Arena Report Content – main points Example Part I

https://code4rena.com/reports/2021-05-fairside

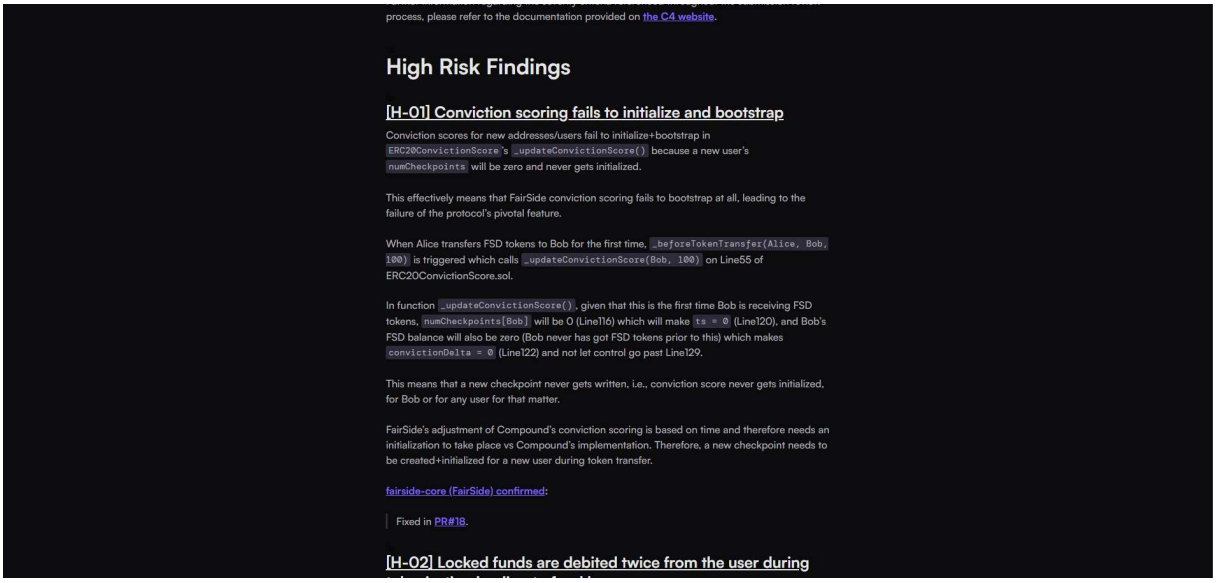


Code4Arena Report - main points Content Example Part II



APPENDIX E

Code4Arena Report - main points Content Example Part III



process, please refer to the documentation provided on the C4 website.

High Risk Findings

[H-01] Conviction scoring fails to initialize and bootstrap

Conviction scores for new addresses/users fail to initialize+bootstrap in `ERC20ConvictionScore`'s `_updateConvictionScore()` because a new user's `numCheckpoints` will be zero and never gets initialized.

This effectively means that FairSide conviction scoring fails to bootstrap at all, leading to the failure of the protocol's pivotal feature.

When Alice transfers FSD tokens to Bob for the first time, `_beforeTokenTransfer(Alice, Bob, 100)` is triggered which calls `_updateConvictionScore(Bob, 100)` on Line55 of ERC20ConvictionScore.sol.

In function `_updateConvictionScore()`, given that this is the first time Bob is receiving FSD tokens, `numCheckpoints[Bob]` will be 0 (Line116) which will make `ts = 0` (Line120), and Bob's FSD balance will also be zero (Bob never has got FSD tokens prior to this) which makes `convictionDelta = 0` (Line122) and not let control go past Line129.

This means that a new checkpoint never gets written, i.e., conviction score never gets initialized, for Bob or for any user for that matter.

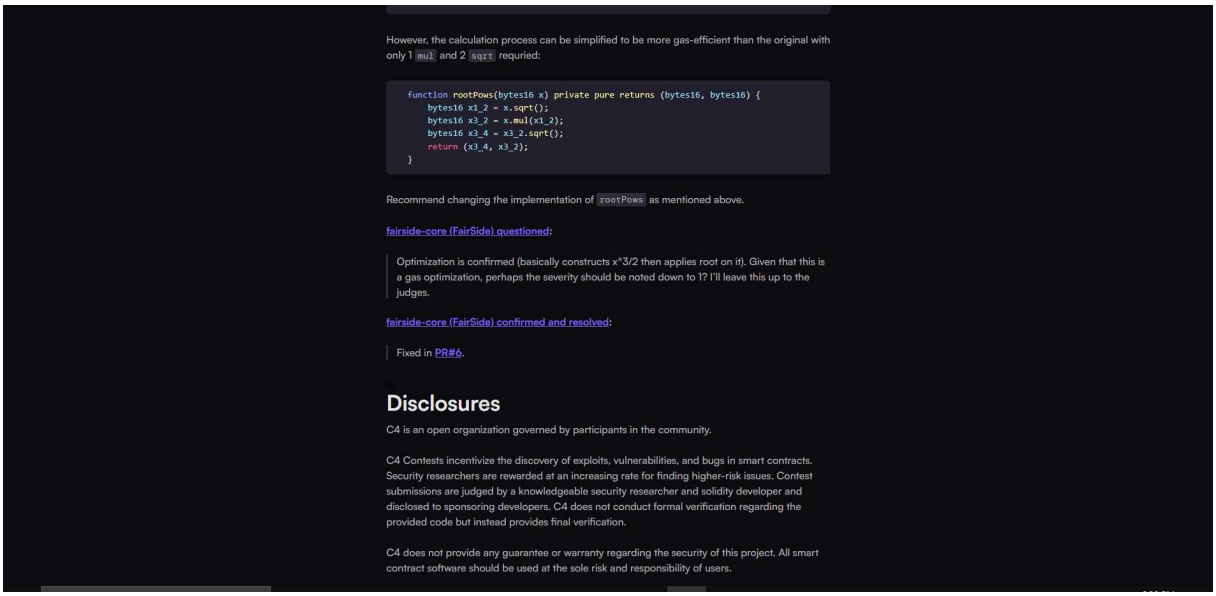
FairSide's adjustment of Compound's conviction scoring is based on time and therefore needs an initialization to take place vs Compound's implementation. Therefore, a new checkpoint needs to be created+initialized for a new user during token transfer.

fairside-core (FairSide) confirmed:

> Fixed in PR#18.

[H-02] Locked funds are debited twice from the user during

Code4Arena Report - main points Content Example Part IV



However, the calculation process can be simplified to be more gas-efficient than the original with only 1 `mul` and 2 `sqrt` requried:

```
function rootPows(bytes16 x) private pure returns (bytes16, bytes16) {
    bytes16 x1_2 = x.sqrt();
    bytes16 x3_2 = x.mul(x1_2);
    bytes16 x3_4 = x3_2.sqrt();
    return (x3_4, x3_2);
}
```

Recommend changing the implementation of `rootPows` as mentioned above.

fairside-core (FairSide) questioned:

> Optimization is confirmed (basically constructs x^3/2 then applies root on it). Given that this is a gas optimization, perhaps the severity should be noted down to 1? I'll leave this up to the judges.

fairside-core (FairSide) confirmed and resolved:

> Fixed in PR#6.

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa