Unveiling the Potential of ChatGPT in Detecting Machine Unauditable Bugs in Smart Contracts: A Preliminary Evaluation and Categorization

Bo Gao, Qingsong Wei, Yong Liu, Rick Siow Mong Goh
*Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR)*
Singapore, Singapore
Email: {gao_bo, wei_qingsong, liuyong, gohsm}@ihpc.a-star.edu.sg

***Abstract*—Smart contracts are becoming an integral part of decentralized applications, yet exploitable bugs in these contracts pose significant threats, often leading to considerable monetary losses. Traditional tools often struggle to identify these bugs, with a recent study indicating that 80% of them are classified as *Machine Unauditable Bugs (MUBs)*, rendering conventional approaches ineffective in addressing such cases. In practice, identifying MUBs requires seasoned expertise and is time-intensive, often stalling project progress. In this work, we present a preliminary evaluation of the performance of ChatGPT, a state-of-the-art large language model, especially in detecting MUBs. Our study first investigates the effectiveness and limitations of ChatGPT in detecting various categories of MUBs with two kinds of prompts, general prompts and guidance prompts, on 246 real-world MUBs collected from Code4rena between 2021 and 2022. Subsequently, we compared the leading tool, SPCON, with ChatGPT on 17 CVE contracts with access control issues (a category of MUBs), and found that ChatGPT exhibited comparable performance but better usability over SPCON. We summarize the implications of our findings for the broader community, shedding light on the model's capabilities, limitations and potentials in detecting smart contract bugs. Our evaluation dataset and results are released at Github[1].**

***Index Terms*—ChatGPT, Exploitable bugs, Effortless usage**

I. Introduction

Smart contracts have revolutionized the blockchain ecosystem by enabling decentralized applications and trustless interactions between parties. The decentralized nature of blockchain platforms, such as Ethereum [1], has spurred the rapid development and deployment of these contracts. However, with the increasing adoption of smart contracts comes the inevitable growth in complexity and value, which subsequently attracts attackers and underscores the importance of bug detection in their development.

A variety of methods have been developed to identify smart contract vulnerabilities, such as reentrancy, integer overflows, Transaction-ordering Dependency (TOD), and Gas-related Issues [2]–[6]. While these approaches have been proven effective in detecting general and relatively simple vulnerabilities across various projects, they are mostly tailored for Machine Auditable Bugs (MABs), i.e., which account for only about 20% of exploitable bugs [7]. Notably, the remaining 80% are categorized as Machine Unauditable Bugs (MUBs), which presents a significant limitation in the applicability of existing tools across a wide range of scenarios. To delve further into this classification, we will elaborate in Section II-B on the distinguishing characteristics between MABs and MUBs.

MUBs pose a considerable challenge in their detection, often demanding substantial manual involvement from experienced practitioners for their identification. In recent times, notable progress in the realm of natural language processing, particularly the development of Large Language Models (LLMs) like ChatGPT [8], offer new possibilities for bug detection and analysis due to their remarkable capabilities. These models stand as a promising pathway for the exploration of alternative strategies aimed at addressing the intricate issues associated with MUBs and potentially complement existing automated tools in smart contract security.

Significant discourse has emerged within the community regarding the effectiveness of LLMs in analyzing smart contracts. Nevertheless, a comprehensive investigation into this matter has remained absent as of this writing. To address this gap, this paper presents a preliminary assessment of ChatGPT's ability in detecting MUBs which we are especially interested in. While improvements to existing tools can enhance the capabilities in detecting MABs, the detection of MUBs requires the expertise of experienced professionals due to their complicate nature. This study first examines 246 instances of real-world MUBs collected from Code4rena between 2021 and 2022, focusing on the model's effectiveness in detecting various MUB categories and understanding its limitations in recognizing different types of MUBs. Furthermore, we compare ChatGPT with the state-of-the-art tool, SPCON [9], in detecting access control problems on 17 contracts reported in Common Vulnerabilities and Exposures (CVE) system [10], and found that ChatGPT exhibits comparable performance with better usability. Moreover, we explore the implications of our findings for the broader community, unveiling the model's capabilities in detecting smart contract bugs and providing insights beneficial to the development of robust LLM-based automated bug detection tools.

To conclude, we make the following contributions.

- A comprehensive assessment of ChatGPT's performance

This Research is supported by the RIE2025 Industry Alignment Fund – Industry Collaboration Project (IAF-ICP) (Award No: I2301E0020), administered by A*STAR

[1]https://github.com/Wormfol/Detect-MUBs-by-ChatGPT/tree/master

```
1 function subscribe(uint posId, uint incentiveId) public {
2     Position memory position = positions[posId];
3     IConcentratedLiquidityPool pool = position.pool;
4     Incentive memory incentive = incentives[pool][posId];
5     Stake storage stake = stakes[posId][incentiveId]; ...
6 }
```

Fig. 1: Example Contract with Implementation Bug

in identifying MUBs across 246 real-world smart contract vulnerabilities.

- A detailed comparative analysis of ChatGPT and SPCON in identifying access control problems in 17 CVE contracts, highlighting the advantages of ChatGPT in terms of usability and effectiveness.
- Insights into the implications of our findings for practitioners and researchers, contributing to the development of more effective automated tools for bug detection and prevention with LLMs.

II. Background and Research Questions

This section introduces a flawed smart contract with implementation bugs, which serves as a basis to discuss bug categorizations. It allows us to define the research questions guiding our study. We assume some familiarity with basic concepts such as blockchain, Ethereum, and smart contracts, and refer readers to [11] for details.

A. Illustrative Example

Fig. 1 shows a smart contract function from the Sushi Trident project by SushiSwap[2] which reveals an implementation bug [12]. This flaw came to light during the Code4rena (C4) contest organized by the Sushi Trident project between September 30 and October 6, 2021. In C4 contests, community members, known as "Wardens," engage in reviewing, auditing, and analyzing smart contract logic. Successful findings reward these Wardens with bounties offered by the host projects.

As shown in Fig. 1, the function *subscribe* is intended to subscribe a certain *Position* identified by **posId** to a certain *Incentive* identified by **incentiveId**. The implementation bug lies in line 4 where the *incentives* mapping is being indexed first by **pool** and then by **posId**. However, **posId** should be **incentiveId**, a counter that increases by one whenever a new incentive is added to the pool. The usage of **posId** could cause the wrong incentive to be used. Such a mistake exemplifies an implementation bug. Detecting it via automated tools can be a formidable challenge since it involves a misunderstanding of the contract's logic, rather than a clear-cut violation of common coding practices or security guidelines.

B. Classification and Categorization of Bugs

This section introduces the different types of bugs that exist within the scope of our study, which also provides the context for our research.

[2]A decentralized exchange and automated market maker built on Ethereum.

1) Description of MABs and MUBs: An exploitable bug is a bug which can cause direct financial loss. Machine Auditable Bugs (MABs) represent exploitable bugs derived from a comprehensive study of 37 automated bug detection methods published in esteemed Software Engineering, Security, and Programming Language conferences and journals between 2017 and 2022. In total, 17 distinct types of MABs are concluded[3]. A bug that aligns with any category within the 17 types is labeled as a **MAB** [7]. In the contrary, any bug that doesn't match any of the MAB categories is categorized as a Machine Unauditable Bug (**MUB**). In our study, MUBs serve as a metric to evaluate the efficacy of automated tools as well as ChatGPT.

2) Categories of MUBs: To differentiate the varying degrees of difficulty in identifying different MUBs and to facilitate the creation of guiding prompts for LLMs, we have organized the MUBs into six distinct categories, they are: (C1) price oracle manipulation; (C2) erroneous accounting; (C3) improper access control; (C4) erroneous state updates; (C5) atomicity violations; and (C6) implementation specific bugs. We refer the interested users to [13] for further information.

C. Key Motivation

Our emphasis on MUBs in this study stems from the following significant observations derived from [7]:

Observation 1: A large portion (80%) of exploitable bugs in the wild are machine unauditable.

Observation 2: Majority of exploitable bugs are difficult to find. Of the detected bugs, 54.29% MUBs were reported by a single auditor, indicating that such bugs would be exploited if this auditor missed, thus, the highest difficulty. Merely 25% of exploitable bugs were identified by three or more auditors.

Observation 3: Different types of MUBs have different auditing difficulties, with price oracle manipulation (C1) and ID uniqueness violation (C3) bugs the hardest (75% by 1 auditor) and the easiest (43% by 1 auditor), respectively.

D. ChatGPT and Prompts

ChatGPT [14] is an advanced large language model developed by OpenAI, based on the GPT-4 [15] architecture. We use GPT-4 and ChatGPT interchangeably hereafter. Trained on a diverse range of internet text, it demonstrates a remarkable ability to generate human-like text and understand context. In the context of smart contract bug detection, ChatGPT's understanding of programming languages, pattern recognition, and knowledge of known vulnerabilities equip it with the potential to identify critical smart contract bugs.

While LLMs, including ChatGPT, have demonstrated remarkable performance across various tasks, their ability to reason also depends heavily on prompt design. Several innovative approaches, such as Chain-of-Thought (CoT) [16], Least-to-Most [17] and Complex CoT [18] have been proposed to bolster the performance of LLMs in datasets demanding

[3]Due to page limitation, we show the table in https://github.com/Wormfol/Detect-MUBs-by-ChatGPT/blob/master/results/bugCat.md#machine-auditable-bugs-mubs

TABLE I: Code4rena Dataset

Reports	MUBs	Relevant files	Contests	LoC (Avg)	Bounties
Code4rena	246	206	80	4675	$5.03M$

reasoning. Taking into account the intrinsic variances across tasks and the dimensions of the inputs, our investigation draws inspiration to come up with different prompts, which are elaborated upon in Section III-B. In our study, we will explore ChatGPT's capabilities in pinpointing MUBs, particularly in the context of varying prompt designs.

E. Research Questions

Based on the above categorization and discussion, we propose the following research questions of particular interest.

1) How effective is ChatGPT, with and without special guidance, in identifying MUBs which elude traditional bug detection tools?
2) How much improvement does guidance prompts provide for ChatGPT compared to general prompts?
3) What are the limitations of ChatGPT in recognizing specific types of MUBs?
4) How does ChatGPT perform comparing with the traditional state-of-the-art tool, SPCON, in identifying one kind of MUBs, access control problems?

III. Evaluation and Key Findings

In this section, we introduce the dataset employed for our evaluation and elaborate on our key discoveries in response to the research questions outlined in Section II-E.

A. Dataset

Our dataset is designed to reflect real-world scenarios and challenges, making it a robust benchmark for evaluation. It comprises of two sets.

The first set, sourced from Zhang et al. [7], consists of 246 MUBs from the Code4rena[4] contest reports. They distributed across 206 contract files from 80 Code4rena contests (projects), spanning from April 2021 to December 2022, as shown in Table I. Each project consists of multiple contract files, and the average total lines of code (LoC (Avg)) without comments and blank for each project is 4675. We skip lengthy contracts surpassing ChatGPT's processing capacity, expecting this constraint to soon be obsolete due to LLM advancements. These bugs represent about \$2.3 billion secured by Code4rena audits and \$5.03 million in bounties, offering a varied dataset for assessing ChatGPT's performance on current vulnerabilities.

The second dataset is selected from 531 smart contract CVEs, exactly the same benchmarks used in [9]. The details are shown in Table IV.

[4]Code4rena [19] is a leading audit contest platform for pre-deployment projects. The platform engages project developers to commit bounties up to \$1M as incentives to draw participants from all over the world. Community experts selected and developers collaboratively review the submitted bug reports and reward the participants based on the severity and frequency of a particular bug submission.

B. Settings

Considering the probabilistic nature of language models such as ChatGPT, which may miss certain issues in a single run, we conduct three runs with identical prompts for each project, collect unique responses, and provide explanations for manual review. Only after this review process is each occurrence of a bug deemed a valid alarm.

Our evaluation makes use of two distinct types of prompts: the general prompt and the guidance prompt. The general prompt is introduced to achieve universal applicability. This means that regardless of the specific smart contract being examined, this prompt can be employed to analyze potential vulnerabilities. By using a broad phrasing, it provides users with a versatile tool that can be applied with minimal effort. The guidance prompt provides a more granular approach to evaluate potential vulnerabilities in smart contracts. Crafted with precision and a distinct focus on specific MUB categories, it guides ChatGPT to analyze those specific areas more deeply. Our initial idea was to offer a concise description of each MUB category within the prompt, similar to the approach adopted in the CoT study [16]. However, given ChatGPT's expansive training data and its inherent understanding of these concepts, we realized that such descriptions were redundant. Our preliminary experiments confirmed this observation, as there was no discernible difference in ChatGPT's responses with or without the category explanations.

One notable trade-off when using the guidance prompt is the need for multiple iterations. Since each prompt is tailored to a specific MUB category, analyzing a smart contract for all possible vulnerabilities requires running the contract with each category-specific prompt. This means that for a smart contract, if there are six MUB categories under consideration, the contract will have to be processed at least six times using ChatGPT – once for each category. This is in contrast to the general prompt, where only a single run is necessary. The following shows the different prompts employed in our evaluation.

- General Prompt: "Identify any general vulnerabilities (like overflow, reentrancy) or the functional vulnerabilities in the following Solidity smart contracts".
- Guidance Prompt: "Analyze the potential for price oracle manipulation in the following smart contract, focusing on how price is calculated or external data is fetched and used in the contract's functions[5]".

While this setup provides an informative overview, more comprehensive results could potentially be achieved by multiple interactive rounds with richer contextual information.

C. ChatGPT's Effectiveness on MUBs with General Prompts

With the general prompt, ChatGPT is able to detect 200 unique high-risk MUB bugs, initially totaling 258 before overlap removal. A comprehensive breakdown of this data

[5]The detailed prompts along with the evaluation results can be found at https://github.com/Wormfol/Detect-MUBs-by-ChatGPT/blob/master/results/prompts

TABLE II: Bug Summary with by ChatGPT General Prompt

GPT_confirm	GPT_in_report	GPT_potential	GPT_incorrect	GPT_alarm
13 (5%)	4	145	38	200 (258)

```
1 function setAddresses(IYETIToken _yeti, IERC20 _yusd)
      external onlyOwner {
2     require(!addressesSet, "addresses already set");
3     yetiToken = _yeti;
4     yusdToken = _yusd;
5     addressesSet = true;
6 }
7 function emergenWithdraw() external {
8     require(msg.sender == Ownable(address(factory)).
          owner(), "Event: caller is ...");    ...
9 }
```

Fig. 2: Alarms by ChatGPT with General Prompt

is provided in Table II. The column "GPT_confirm" refers to MUB bugs which feature in Code4rena's reports and are also detected by ChatGPT. A closer inspection of the figures reveals that merely 13 bugs, or 5% of the total bugs from the Code4rena's report, were confirmed as MUBs by ChatGPT.

Finding 1: ChatGPT demonstrates limited effectiveness (5% averagely) in detecting MUBs with general prompts.

We proceeded to analyze the remaining 187 alarms raised by ChatGPT in Table II, which are categorized as "GPT_in_report" (4), "GPT_potential" (145) and "GPT_incorrect" (38). The "GPT_in_report" category includes MUB bugs identified by ChatGPT that feature in the Code4rena reports but aren't considered high-risk by the Code4rena community. These findings, however, still bear significance as such classification often hinges on the developers' subjective discretion.

"GPT_potential" refers to potential bugs that may serve as alerts for developers but are not strictly problematic. For example, in Fig. 2, the function $setAddresses$ sets the addresses for the YETI and yUSD tokens by the owner. While ChatGPT correctly comprehends the intent of this function, it still raises an alarm because it thinks "The function can only be called once. If the owner forgets to set the addresses, the contract could become unusable". Although we regard this explanation as overly cautious, we categorize such alarms as potential bugs since they could remind users to exercise caution, even if the likelihood of occurrence is very low. "GPT_incorrect" includes instances that were proved false after manual verification. This category captures various types of ChatGPT errors, which are:

1) ChatGPT sometimes neglects existing constraints within functions and raises unwarranted alarms. For instance, in Fig. 2, it incorrectly suggests $emergenWithdraw$ could be called by any external address, despite line 8 managing this issue.
2) ChatGPT does not capture the functions correctly, issuing warnings for non-existent variables or issues.
3) ChatGPT does not always understand the basic principles of smart contracts, such as the fact that *uint* type cannot be negative, that overflow is automatically checked from Solidity 0.8.0 onwards, etc.

TABLE III: Bugs by ChatGPT with Guidance Prompts

Bug Cat.	C1	C2	C3	C4	C5	C6	Total
Code4rena	24	63	34	21	46	59	246
GPT_alarm	63	148	76	46	192	105	630
-GPT_confirm	8	5	9	7	5	2	36
-GPT_in_report	5	7	7	2	6	5	32
-GPT_potential	13	55	44	13	105	69	299
-GPT_incorrect	37	81	16	24	76	29	263
GPT_conf(%)	33%	8%	26%	33%	11%	3%	15%

TABLE IV: SPCON vs ChatGPT on CVE

CVE Report	LOC	SPCON	GPT			
			confirm	newb	potential	incorrect
CVE-2018-10666	163	✓	✓	0	2	0
CVE-2018-10705	98	✓	✓	0	1	0
CVE-2018-11329	113	✓	✗	0	2	0
CVE-2018-17111	72	✗	✓	0	1	0
CVE-2018-19830	140	N/A1	✗	0	2	3
CVE-2018-19831	233	✓	✓	1	2	2
CVE-2018-19832	174	✓	✗	0	2	2
CVE-2018-19833	60	N/A1	✓	0	1	0
CVE-2018-19834	139	N/A1	✓	0	2	2
CVE-2019-15078	174	✓	✗	0	3	2
CVE-2019-15079	58	✓	✗	0	1	1
CVE-2019-15080	121	✓	✓	0	2	1
CVE-2020-17753	350	✗	✗	0	5	2
CVE-2020-35962	540	✗	✓	0	3	1
CVE-2021-3006	120	N/A2	N/A2	N/A2	N/A2	N/A2
CVE-2021-34272	133	✓	✗	0	2	1
CVE-2021-34273	80	✗	✓	1	1	0

Finding 2: While ChatGPT comprehends most principles and rules of smart contracts, it occasionally demonstrates inconsistent understanding by raising false alarms.

In addition to ChatGPT's inconsistent understanding of smart contracts, the discrepancy in bug counts between Code4rena and ChatGPT primarily stems from two factors: (1) the model's limited capacity to identify high-risk bugs with general prompts and; (2) the token (input length) restriction that usually confines the analysis to a single file.

D. *ChatGPT's Effectiveness on MUBs with Guidance Prompts*

To evaluate the effect of guidance prompts on ChatGPT's bug detection capabilities, we conducted experiments on the six MUB categories' bugs. Table III presents the results, demonstrating the performance of ChatGPT when using category-specific prompts. The "Code4rena" row records the count of high-risk bugs from the Code4rena reports per category. The column headers parallel with the ones explained in Table II and will not be reiterated here. Importantly, "GPT_conf(%)" calculates the ratio of bugs confirmed to the total bugs present in the Code4rena reports, offering an insight into the model's precision for each category. The "Total" column represents the aggregate of bugs reported in Code4rena and detected by ChatGPT across all categories.

Table III reveals that, with guidance prompts, ChatGPT detected a total of 36 bugs (15% of the bugs in Code4rena reports). This is a noticeable increase from the 13 bugs (5%) it was able to detect with general prompts, as detailed in section III-C. This improvement underscores the significance of crafting specific and detailed prompts to maximize ChatGPT's potential in bug detection tasks.

Finding 3: Guidance prompts enhance ChatGPT's effectiveness in identifying bugs, raising detection rate to 15% on average, compared to 5% with general prompts.

The table further showcases that the effectiveness of ChatGPT in identifying high-risk bugs varies across different bug categories, as shown in the "GPT_conf(%)" row. ChatGPT identified 33% of high-risk bugs in price oracle vulnerabilities (C1) and erroneous state updates (C4), but only detected 3% in specific implementation bugs (C6). This discrepancy likely stems from two factors: 1) the more conceptual or structural vulnerabilities, like C1 and C4, may be easier to identify through code analysis and pattern recognition; 2) ChatGPT's performance is largely dependent on the data it was trained on. It's likely that the data includes more examples and discussions related to vulnerabilities C1 and C4. Conversely, specific implementation bugs (C6) which vary significantly and lack generic patterns, are less represented in the training data, affecting the ChatGPT's capability to identify them effectively.

Finding 4: ChatGPT's performance varies across different bug categories, showing differential proficiency in recognizing high-risk bugs.

E. ChatGPT vs SPCON on Finding Access Control (Permission) Bugs

Being one category of the significant MUBs, access control bugs has attracted some research attention. To the best of our knowledge, only one study, SPCON [9], which significantly outperforms pattern-based tools like Slither [20], Oyente [21], Maian [22], Manticore [4], and SmartCheck [23], has been conducted. This study utilizes specification-validation methods to mine past transactions of a contract, recover a probable access control model, and subsequently check it against various information flow policies to identify potential user permission-related bugs. This section aims to compare the effectiveness of ChatGPT and SPCON in identifying one of MUBs, access control vulnerabilities.

To carry out a comparative analysis, we conducted experiments on the same 17 CVE contracts as analyzed by SPCON. The experiment prompt for ChatGPT was: "Please identify inappropriate access control issues like privilege escalation, unrestricted function calls, and ID-related violations vulnerabilities in the following smart contract." The results are presented in Table IV. Additionally, we excluded CVE-2021-3006, labeled as "N/A2," along with three other CVEs labeled as "N/A1" by the SPCON author. The contract corresponding to CVE-2021-3006 did not include the vulnerable function reported in [24], and we could not locate any relevant contract site-wide. We suspect that the relevant contract was already destructed. In the remaining 16 contracts, both SPCON and ChatGPT successfully identified 9 bugs. Interestingly, ChatGPT also identified two additional new bugs that were overlooked in the CVE report and the study by Liu et al [9]. As before, ChatGPT raised 32 alarms, which were classified as potential bugs. Concurrently, ChatGPT generated 17 incorrect alarms, which aligns with **Finding 2** in SectionIII-C.

Finding 5: ChatGPT exhibits comparable performance to the SOTA, SPCON, in detecting access control problems.

Additionally, efforts were made to assess whether the effectiveness of traditional tools like SPCON lose effectiveness as smart contract size increases, while the advantage of ChatGPT amplifies. In the comparative evaluation, the largest files are only 350 and 540 lines, respectively. These sizes are relatively insignificant when compared to contemporary decentralized applications (DApps) and are also considerably smaller than the projects discussed, which is 4675 as in Table I. However, SPCON's evaluation was hindered by its dependency on actual transactions, which were absent due to projects not being deployed. In conclusion, ChatGPT demonstrates better usability with comparable performance to SPCON in detecting MUBs, specifically access control problems.

This conclusion supports the notion that LLMs like ChatGPT are not only capable of providing comparable performance to specialized tools like SPCON in detecting access control problems but may also offer better usability and scalability as the size and complexity of smart contracts increase. This indicates a promising direction for future research and development in the field of smart contract security analysis.

IV. Related Work

Extensive research has been conducted to evaluate the efficacy of conventional automated tools [20]–[22], [25]–[27] relying on techniques such as symbolic execution, fuzzing, and formal verification, with their assessment benchmarks primarily targeting machine auditable bugs (MABs) like overflow, underflow, re-entrancy, and transaction ordering dependency, among others. Machine learning methods have also been employed for bug detection in smart contracts, with notable projects such as ContractWard [28], EtherGIS [29], AME-VulDetector [23], and ESCORT [30]. While these tools have been effective in detecting MABs, they struggle to identify MUBs, which are more complex and challenging to detect. Certain tools like [20]–[22], [27] have the capability to address specific types of MUBs, such as access control bugs, which include suicidal contracts that can be killed by anyone or prodigal contracts that can send Ether to anyone. However, these tools lack comprehensiveness and efficiency.

MUBs represent a category of vulnerabilities which often involve complex logic and require a deep understanding of the smart contract's functionality and the underlying blockchain platform. As a result, the identification of MUBs often necessitates the expertise of seasoned professionals and is time-consuming, thereby impeding the progress of numerous projects. Despite the challenges associated with MUBs, some efforts have been made to develop automated tools for their detection. For example, SPCON [9] is a specification-validation method that mines past transactions of a contract to recover a likely access control model, which can then be checked against various information flow policies to identify potential bugs related to user permissions. However, this approach has its limitations, as it depends on real transactions to acquire roles, which may not always be available, particularly when the contracts are under development.

In recent years, large language models (LLMs) have emerged as a potential solution for detecting MUBs. LLMs, such as ChatGPT, are trained on vast amounts of text data and have the ability to understand and generate human-like text. This enables them to analyze smart contracts and identify potential vulnerabilities that may elude traditional automated tools. To the best of our knowledge, our work is the first to evaluate ChatGPT's capabilities concerning machine unauditable bugs (MUBs), and to compare it with the state-of-the-art tool, SPCON, in detecting access control problems, providing a pioneering exploration into automating the detection of these complex and challenging bugs with large language models (LLMs)

V. Conclusion and Future Work

In this study, we assessed ChatGPT's capability in detecting MUBs in smart contracts. Although the results demonstrated a promising improvement, with an average detection rate of 15% using guidance prompts compared to a 5% detection rate with general prompts, there is an urgent need for further advancement to assist human auditors in tackling challenging tasks, such as price oracle manipulation. The following are the proposed directions for future research:

1) **Prompt Optimization:** Large language models (LLMs) like ChatGPT are highly sensitive to prompts. Future research could focus on generating effective prompts to guide these models more efficiently. Moreover, refining categorization methodologies for different bugs could prevent repetitive runs of the same smart contract.
2) **Domain-Specific Fine-Tuning:** While ChatGPT has a general understanding of smart contracts, it tends to make naive mistakes. Therefore, domain-specific fine-tuning could probably improve the detection accuracy.
3) **Balancing Accuracy and Creativity:** An emphasis on accuracy might lead to higher certainty but lower creativity, limiting the model's ability to detect novel bugs. A careful balance between these aspects is necessary for maximizing the potential of LLMs in this field.

In conclusion, LLMs like ChatGPT represent a promising avenue towards automating the detection of intricate and challenging vulnerabilities, thereby reducing the burden on human auditors. However, realizing this potential will necessitate ongoing research and innovative advancements in the fields of machine learning and smart contract security. This study represents a first step in that direction, providing valuable insights that can inform future efforts to develop more effective and efficient automated tools for bug detection and prevention.

References

[1] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[2] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," pp. 254–269, 2016.

[3] J. Chang, B. Gao, H. Xiao, and J. Sun, "scompile: Critical path identification and analysis for smart contracts," pp. 286–304, 2019.

[4] M. Mossberg *et al.*, "Manticore: A user-friendly symbolic execution framework for binaries and smart contracts," pp. 1186–1189, 2019.

[5] S. So, M. Lee, J. Park, H. Lee, and H. Oh, "Verismart: A highly precise safety verifier for ethereum smart contracts," pp. 1678–1694, 2020.

[6] B. Gao, L. Shi, J. Li *et al.*, "sverify: Verifying smart contracts through lazy annotation and learning," pp. 453–469, 2021.

[7] Z. Zhang, B. Zhang, W. Xu, and Z. Lin, "Demystifying exploitable bugs in smart contracts," 2023.

[8] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[9] Y. Liu, Y. Li, S.-W. Lin, and C. Artho, "Finding permission bugs in smart contracts with role mining," pp. 716–727, 2022.

[10] CVE. (2023) Cve program. Accessed: 2023-08-20. [Online]. Available: https://www.cve.org/

[11] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum yellow paper*, vol. 151, no. 2014, 2014.

[12] Code4rena. (2023) Sushi trident contest. Accessed: 2023-04-20. [Online]. Available: https://code4rena.com/reports/2021-09-sushitrident-2#h-02-wrong-usage-of-positionid-in-concentratedliquiditypoolmanager

[13] Wormfol. (2023) Bug categorization. Accessed: 2023-05-26. [Online]. Available: https://github.com/Wormfol/Detect-MUBs-by-ChatGPT/blob/main/results/bugCat.md#bug-categorization

[14] OpenAI. (2023) Chatgpt. Accessed: 2023-04-16. [Online]. Available: https://openai.com/

[15] ——, "Gpt-4 technical report," 2023.

[16] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[17] D. Zhou *et al.*, "Least-to-most prompting enables complex reasoning in large language models," *arXiv preprint arXiv:2205.10625*, 2022.

[18] Y. Fu *et al.*, "Complexity-based prompting for multi-step reasoning," *arXiv preprint arXiv:2210.00720*, 2022.

[19] Code4rena. (2023) Code4rena contest platform. Accessed: 2023-04-16. [Online]. Available: https://code4rena.com/

[20] J. Feist, G. Grieco, and A. Groce, "Slither: a static analysis framework for smart contracts," pp. 8–15, 2019.

[21] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," pp. 254–269, 2016.

[22] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," pp. 653–663, 2018.

[23] Z. Liu *et al.*, "Smart contract vulnerability detection: from pure neural network to interpretable graph feature and expert pattern fusion," *arXiv preprint arXiv:2106.09282*, 2021.

[24] CVE. (2023) Cve-2021-3006 detail. Accessed: 2023-08-20. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-3006

[25] T. Durieux *et al.*, "Empirical review of automated analysis tools on 47,587 ethereum smart contracts," pp. 530–541, 2020.

[26] M. Ren, Z. Yin, F. Ma *et al.*, "Empirical evaluation of smart contract testing: What is the best choice?" pp. 566–579, 2021.

[27] S. Tikhomirov *et al.*, "Smartcheck: Static analysis of ethereum smart contracts," pp. 9–16, 2018.

[28] W. Wang *et al.*, "Contractward: Automated vulnerability detection models for ethereum smart contracts," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1133–1144, 2020.

[29] Q. Zeng *et al.*, "Ethergis: A vulnerability detection framework for ethereum smart contracts based on graph learning features," pp. 1742–1749, 2022.

[30] C. Sendner *et al.*, "Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning."