To thoroughly examine our work's performance, we adopt GPTScan [2] as one benchmark for comparison. In this document, we explain the detailed implementation of extending GPTScan to identify the Reentrancy (REE) vulnerability.

1 Basic information of GPTScan

GPTScan does not explicitly involve the detection of REE attacks. However, it provides a general framework for detecting logical vulnerabilities by utilizing the Generative Pre-trained Transformer (GPT) and static analysis. Specifically, GPTScan initiates the process by applying multi-dimensional filtering and reachability analysis to determine candidate functions that potentially have flaws. Subsequently, GPTScan extracts the characteristics of vulnerable functions from historical and typical cases, mapping functional ones to *scenarios* and operational ones to *properties*. These characteristics are used for code understanding to match potential vulnerabilities, enabling semantic-level comparison. Additionally, to mitigate GPT's inherent limitations in processing syntax details, GPTScan employs GPT to extract key variables and statements while using a static analysis tool for fine-grained validation.

2 Extension GPTScan for REE Detection

2.1 Prompt Design of Scenario and Property Matching

For scenario and property matching, GPTScan utilizes GPT-4 to generate preliminary descriptions of scenarios and properties from previous vulnerability reports. These descriptions are iteratively updated on original datasets until they allow the known vulnerabilities to pass. Following a similar procedure, we first adopted GPT-4-generated prompts tailored to REE vulnerabilities. To improve performance, iterative adjustment of the prompt was made using the SmartBugs [1] and Rudder [3] datasets. After rounds of iterations, the final prompt is defined in Table 1.

Prompt	Description
Scenario	have an external call
Property	and followed by any state changes

Table 1. The Prompt of Scenario and Property Matching

As shown in Table 2, the corresponding metrics exhibit good robustness and effectiveness of our prompt. Despite minimal false positive cases, we still achieved a precision of 1 and a recall of 96.7% in the SmartBugs. Meanwhile, in the Rudder, we also obtained a precision of 95.5% and a recall of 1.

Dataset	TP	TN	FP	FN	Precision	Recall	FPR
SmartBugs	30	0	0	1	1	0.967	0
Rudder	42	0	2	0	0.955	1	1

Table 2. Analyse the effectiveness of our prompt on two datasets

2.2 Final Design of Static Analysis and GPT-assisted Validation

GPTScan provides four primary static analysis methods for further validation. Combined with the typical characteristics of REE, we choose the Order Check (OC) and Function Call Argument Check

(FA) methods to perform static analysis. OC assists us with checking the order of state variable updates and external calls, while FA supports us in confirming whether the user of the external call is trusted. However, using Falcon as described in GPTScan caused compilation issues in static analysis. To address this, we reimplemented both checks based on Slither.

Therefore, the corresponding prompt in this step is to extract statements where the state variable updates, the external call happens, and the variable related to the external caller. Our prompt is:

In this function, which statement updates a state variable? Please answer in a section starting with "StatementsA:".
In this function, which statement executes an external call? Please answer in a section starting with "StatementsB:".
In this function, which variable holds the sender's address of the external call? Please answer in a section starting with "VariableA:".
Please answer strictly in the following format: "StatementsA": "State update statement":"Description", "StatementsB": "External call statement":"Description", "VariableA": "Variable name": "Description"

Table 3. The Prompt of GPT Recognition

References

[1] Thomas Durieux, João F Ferreira, Rui Abreu, and Pedro Cruz. 2020. Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In *Proceedings of the ACM/IEEE 42nd International conference on software engineering*. 530–541.

[2] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, and Yang Liu. 2024. Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[3] Zibin Zheng, Neng Zhang, Jianzhong Su, Zhijie Zhong, Mingxi Ye, and Jiachi Chen. 2023. Turn the rudder: A beacon of reentrancy detection for smart contracts on ethereum. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 295–306.