GeekCon

SOLA-FREE

# Transforming Language Models into Smart Contract Audit Experts


TEAM:

- Yuqiang Sun (NTU)
- Daoyuan Wu (HKUST)

# 01

# Introduction

Why traditional static analysis tools failed on smart contracts?

# Prevalence of vulnerabilities on smart contracts

More than **80%** of the exploitable vulnerability for smart contracts are **machine undetectable**.

Most of them are related to the **business logic**.

# Example for Logic Bugs

The first depositor could get all the shares and manipulate the price per share

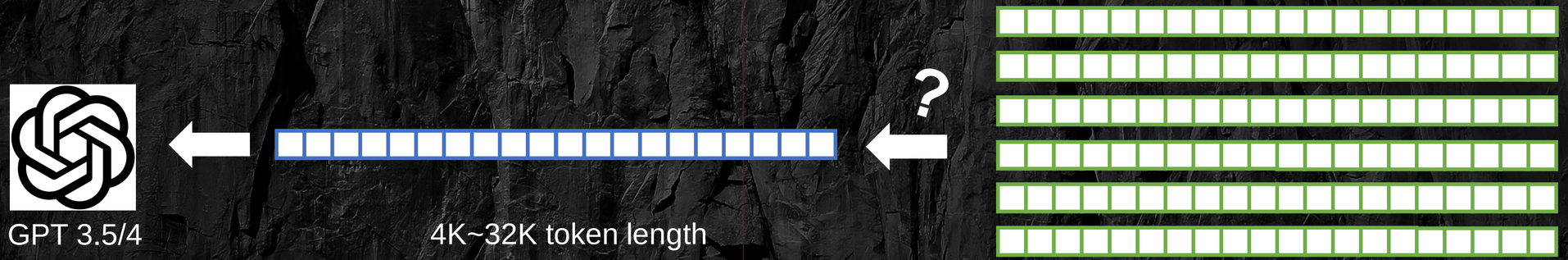
To detect the vuln in the example:

1. Know it is a deposit
2. Find the share calculation statement
3. Check the if branch

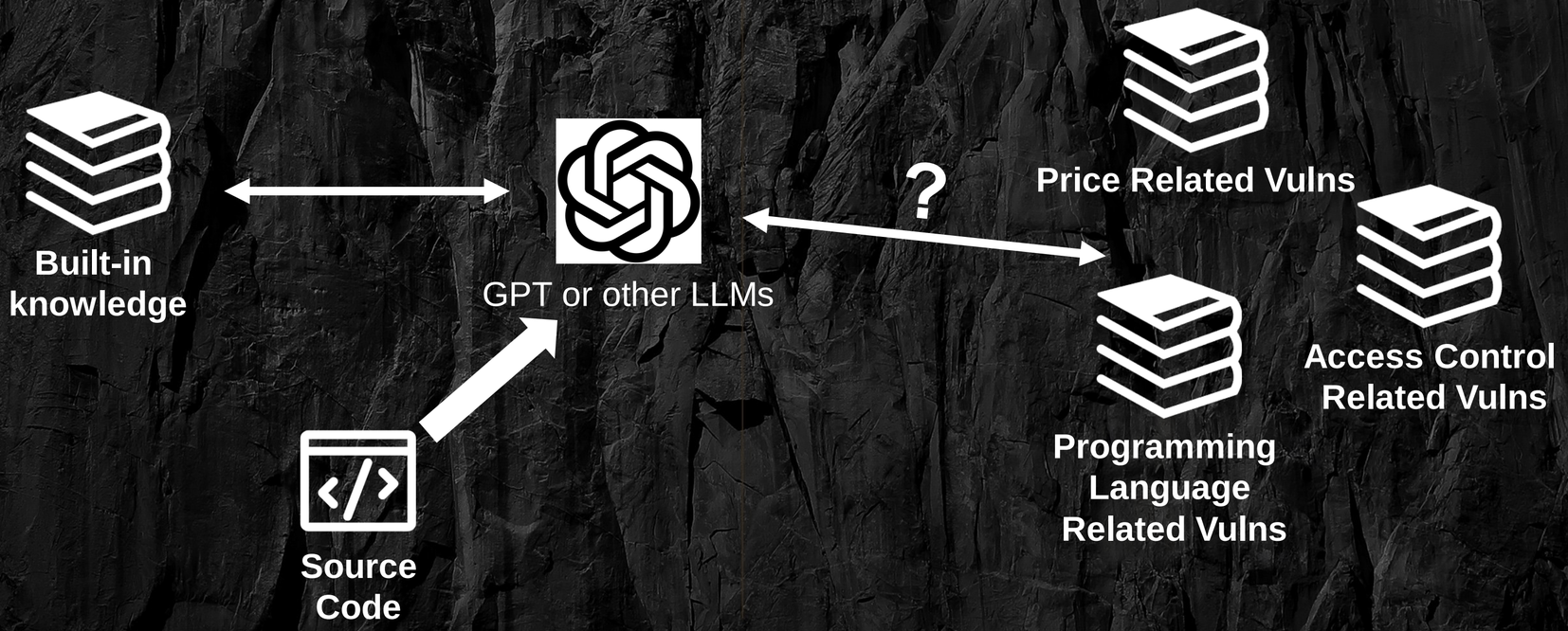
```
function deposit(uint256 _amount) external returns (uint256) {
    uint256 _pool = balance();
    uint256 _before = token.balanceOf(address(this));
    token.safeTransferFrom(msg.sender, address(this), _amount);
    uint256 _after = token.balanceOf(address(this));
    _amount = _after.sub(_before); // Additional check for deflationary
        tokens
    uint256 _shares = 0;
    if (totalSupply() == 0) {
        _shares = _amount;
    } else {
        _shares = (_amount.mul(totalSupply())).div(_pool);
    }
    _mint(msg.sender, _shares);
}
```

# Solution?

# Challenges 1:
# Limited input length



# Challenges 2:
# Domain Specific Knowledge



# Other Challenges

## Too complex tasks

LLM may not be able to understand tasks that are too complex.

## Hallucination

LLMs may have hallucinations and will not always give the correct answer.

## Other problems

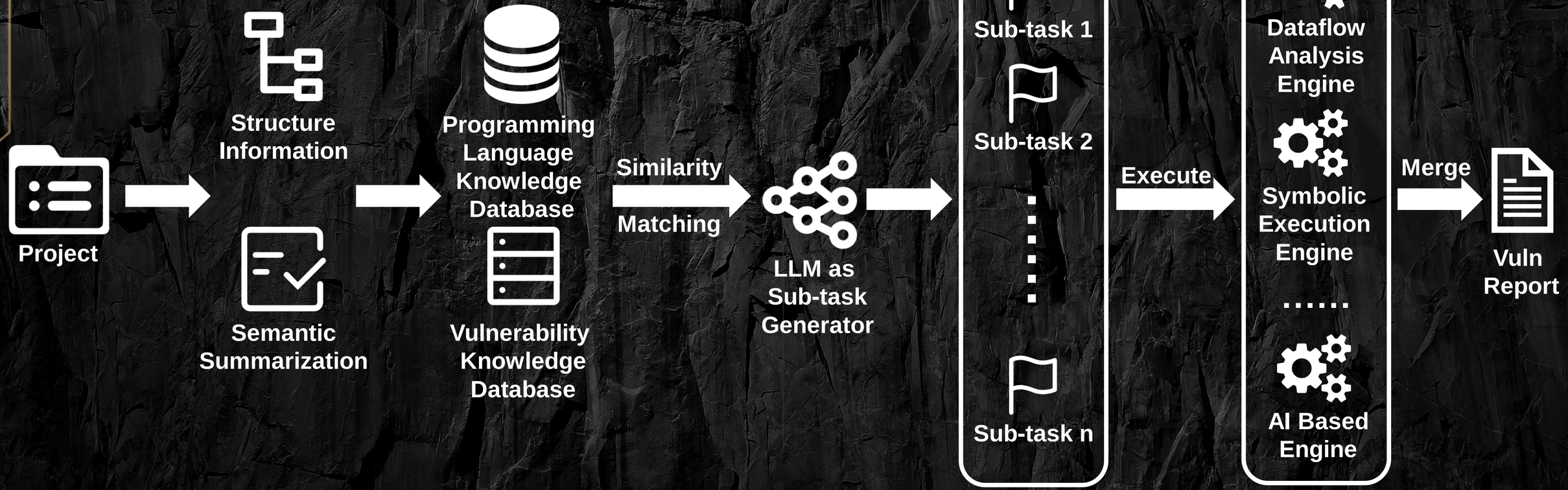
Lack of proper training data, and other challenges.

# 02

# Method

How LLM could help detect such logic vulnerabilities?

# AuditGPT


Project
Structure Information
Semantic Summarization
Programming Language Knowledge Database
Vulnerability Knowledge Database
Similarity Matching
LLM as Sub-task Generator
Sub-task 1
Sub-task 2
Sub-task n
Execute
Dataflow Analysis Engine
Symbolic Execution Engine
AI Based Engine
Merge
Vuln Report

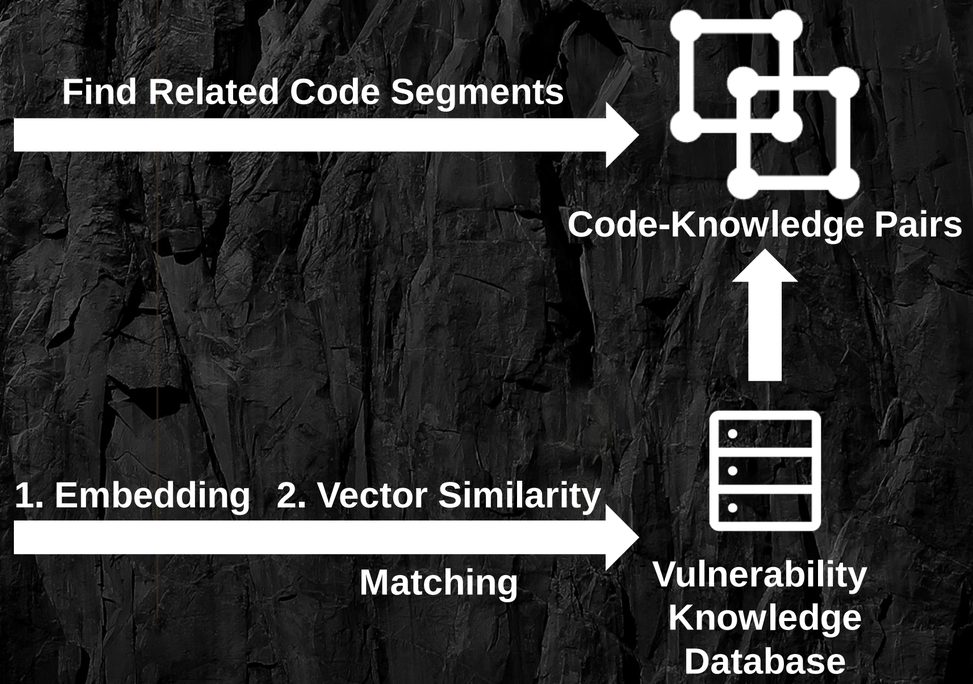
# Step 1
# Code Summarization & Knowledge Matching

## Structural Information

- Call graph
- Usage of state variables
- Class inheritance
- Control flow
- …

## Semantic Information

- Functionality of projects, contracts and functions
- Business model of the project



# Step 2
# Task Decomposing

**Complex task**

Neither handle by LLM or tools

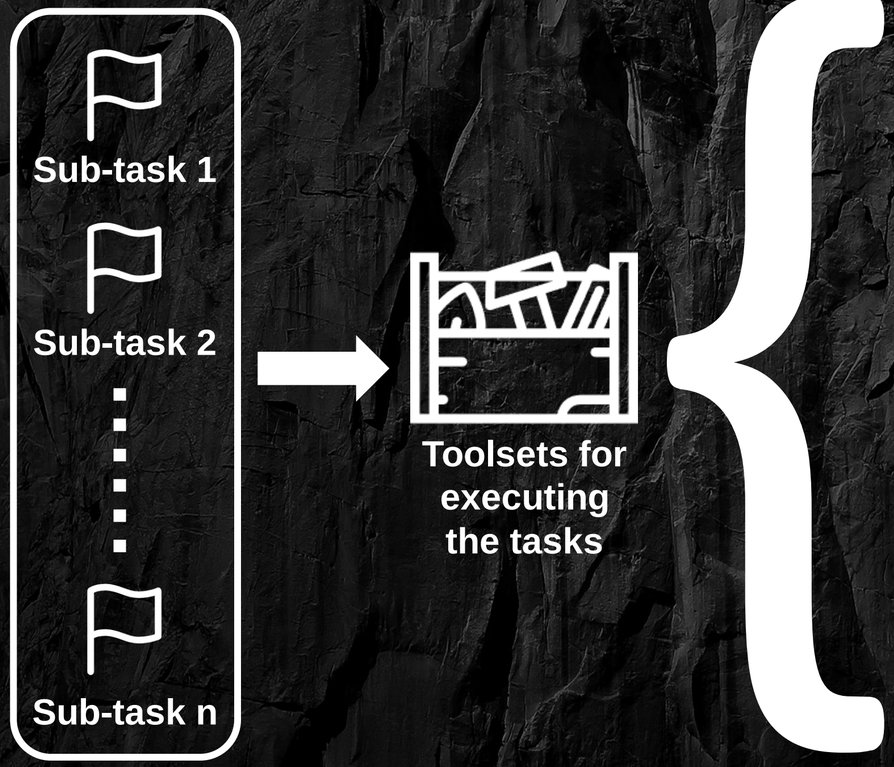
**Decompose**


**Simple Tasks**

Can be handled by LLM or static analysis tools

```
[23:26:35] Running the analyzer                                    analyzer.py:17
                              Retrieved Knowledge
When a pool allows the first depositor to lock an extremely small amount of liquidity, it can be manipulated to cause undesired outcomes
for future liquidity providers, such as rounding down their minted shares to zero. By transferring a large amount of attributions to the
pool after depositing a small amount, an attacker can obtain higher shares of the pool than they should. When the second provider
deposits into the pool, the amount they get may round down to zero due to the small initial liquidity provided, effectively allowing the
attacker to steal funds from other liquidity providers. The core vulnerability lies in the improper handling of the smallest possible
amount deposited and attributions during the calculation of new liquidity provider shares. By exploiting this vulnerability, attackers
can cause asset loss or compromise the pool's integrity.
                                 Current Step
Does the deposit function calculate shares without checking for a minimum deposit amount?
```

# Step 3
# Sub-task Engine




Current Step

Does the deposit function calculate shares without checking for a minimum deposit amount?

*Dataflow Analysis*

| Source | Sink |
|---|---|
| _amount | _shares |

[23:26:50] Step **Dataflow** finished with result *True* step.py:16

Dataflow Analysis Engine

Current Step

Does the function allow the deposit of extremely small amounts which can lead to extremely small or zero share allocation for future liquidity providers?

*Symbolic Execution*

| Variable | Boundary |
|---|---|
| _shares | 1 ~ 1000000000000 |

[23:26:55] Step **Symbolic Execution** finished with result *True* step.py:16

Symbolic Execution Engine

Single Choice Scenario

**Knowledge:** have inside code statements that update/accrue interest/exchange rate, and have inside code statements that calculate/assign/distribute the balance/share/stake/fee/loan/reward

**Code:**

```
    function pendingReward(address _user)
        external
        view
        returns (uint256)
    {
        uint256 rewardShare = accRewardPerShare;
        uint256 staked = totalStaked;
        //if blockNumber is greater than the pool's `lastRewardBlock` the pool's `accRewardPerShare` is outdated,
        //we need to calculate the up to date amount to return an accurate reward value
        if (block.number > lastRewardBlock && staked > 0) {
            (rewardShare, ) = _computeUpdate();
        }
        return
            //rewards that the user had already accumulated but not claimed
            userPendingRewards[_user] +
            //subtracting the user's `lastAccRewardPerShare` from the pool's `accRewardPerShare` results in the amount of rewards per share
            //the pool has accumulated since the user's last claim, multiplying it by the user's shares results in the amount of new rewards claimable
            //by the user
            (balanceOf[_user] * (rewardShare - userLastAccRewardPerShare[_user])) /
            1e36;
    }
```

Response

Yes.

AI Based Engine

# 03

# Impact

How was the impact of AuditGPT in real world?

# Vulnerability Bounties

Secure3

**32** new vulnerabilities

code4rena

**9** new vulnerabilities

TOTAL

**30+**

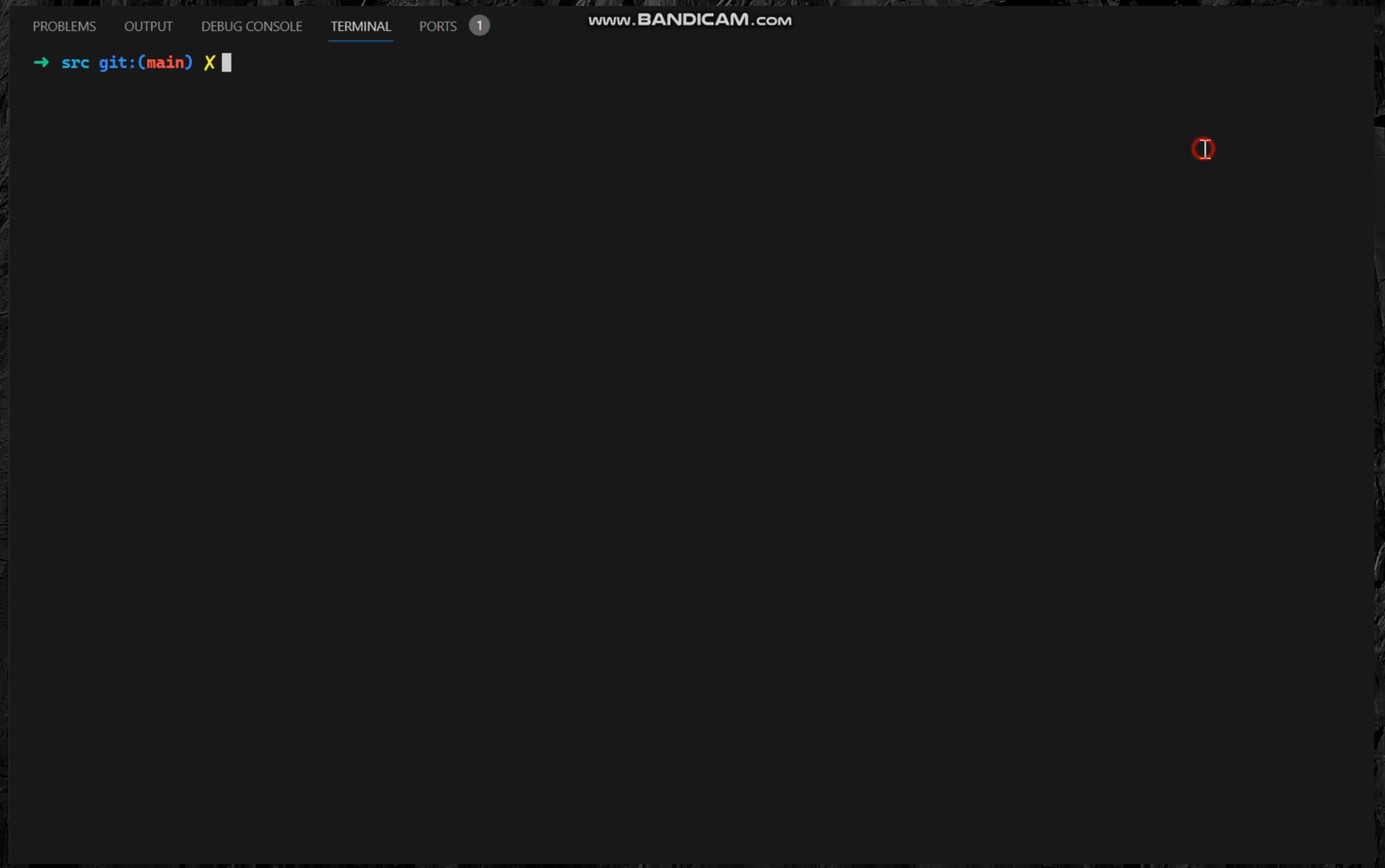
projects

TOTAL

**10k**

bounty per month

# Demo


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
www.BANDICAM.com
➜ src git:(main) X

# Related Studies

AuditGPT is based on the following works:

1. GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis (ICSE 2024)
2. LLM4Vuln: A Unified Evaluation Framework for Decoupling and Enhancing LLMs' Vulnerability Reasoning (arXiv:2401.16185)
3. PropertyGPT: LLM-driven Formal Verification of Smart Contracts through Retrieval-Augmented Property Generation (arXiv:2405.02580)

GeekCon

# THANKS