GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis

Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, Yang Liu

Nanyang Technological University

MetaTrust Labs

East China Normal University

Xi’an Jiaotong University

Singapore Management University

Background

Vulnerability detection for smart contracts

- Smart contracts are programs running on block chains
- They usually provide **financial services**
- Attacks on smart contracts has caused more than **$1,000,000,000** loss
- More than **80%** of the exploitable bugs are machine undetectable
- The reason is that most of them are business logic related

Example

How to detect 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 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);
}
```

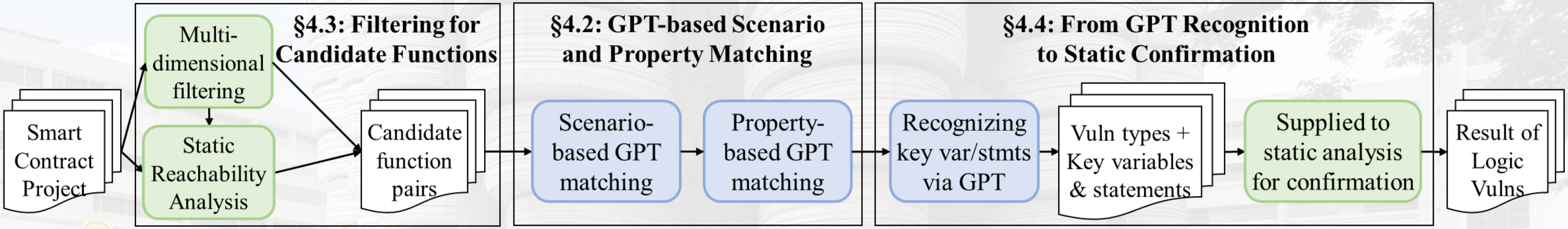
Challenges

1. There are too much code for LLMs to read in a project
2. It's hard to understand the functionality of the given code
3. LLMs may not always give the correct answer

Method Overview

The **FISRT tool** on logic bug detection for smart contracts

- 1. Filtering for candidate code segments
- 2. Scenario and property matches
- 3. Static analysis-based confirmation



Method

Step 1: Filtering

Text-based Filter			
Function Name	Function Content	Access Control	Call Chain

Whitelist-based Filter

Signatures of commonly used libraries from OpenZeppelin

Reachability Analysis

Use call graph to find reachable functions

Method

Step 2: Scenario and property matching

NANYANG TECHNOLOGICAL UNIVERSITY

- Scenario Matching
  - Multiple-choice questions
  - Matching the functionality of the code
- Property Matching
  - Yes/No questions
  - Matching the root cause of the vulnerabilities

Scenario Matching

System: You are a smart contract auditor. You will be asked questions related to code properties. You can mimic answering them in the background five times and provide me with the most frequently appearing answer. Furthermore, please strictly adhere to the output format specified in the question; there is no need to explain your answer.

Given the following smart contract code, answer the questions below and organize the result in a json format like {"1": "Yes" or "No", "2": "Yes" or "No"}.

"1": [%SCENARIO_1%]?
"2": [%SCENARIO_2%]?

[%CODE%]

Property Matching

System: You are a smart contract auditor. You will be asked questions related to code properties. You can mimic answering them in the background five times and provide me with the most frequently appearing answer. Furthermore, please strictly adhere to the output format specified in the question; there is no need to explain your answer.

Does the following smart contract code "[%SCENARIO+PROPERTY%]"? Answer only "Yes" or "No".

[%CODE%]

Method
Rules

Vulnerability Type	Scenario and Property	Filtering Type	Static Check
Approval Not Cleared	**Scenario:** add or check approval via require/if statements before the token transfer **Property:** and there is no clear/reset of the approval when the transfer finishes its main branch or encounters exceptions	FNI, FCCE	VC
Risky First Deposit	**Scenario:** deposit/mint/add the liquidity pool/amount/share **Property:** and set the total share to the number of first deposit when the supply/liquidity is 0	FCCE	DF, VC
Price Manipulation by AMM	**Scenario:** have code statements that get or calculate LP token's value/price **Property:** based on the market reserves/AMMprice/exchangeRate OR the custom token balanceOf/totalSupply/amount/liquidity calculation	FNK, FCCE	DF
Price Manipulation by Buying Tokens	**Scenario:** buy some tokens **Property:** using Uniswap/PancakeSwap APIs	FNK, FCE	FA
Vote Manipulation by Flashloan	**Scenario:** calculate vote amount/number **Property:** and this vote amount/number is from a vote weight that might be manipulated by flashloan	FCCE	DF
Front Running	**Scenario:** mint or vest or collect token/liquidity/earning and assign them to the address recipient or to variable **Property:** and this operation could be front run to benefit the account/address that can be controlled by the parameter and has no sender check in the function code	FNK, FPNC, FPT, FCNE, FNM	FA
Wrong Interest Rate Order	**Scenario:** have inside code statements that update/accrue interest/exchange rate **Property:** and have inside code statements that calculate/assign/distribute the balance/share/stake/fee/loan/reward	FCE, CEN	OC
Wrong Checkpoint Order	**Scenario:** have inside code statements that invoke user checkpoint **Property:** and have inside code statements that calculate/assign/distribute the balance/share/stake/fee/loan/reward	FCE, CEN	OC
Slippage	**Scenario:** involve calculating swap/liquidity or adding liquidity, and there is asset exchanges or price queries **Property:** but this operation could be attacked by Slippage/Sandwich Attack due to no slip limit/minimum value check	FCCE, FCNCE	VC
Unauthorized Transfer	**Scenario:** involve transfering token from an address different from message sender **Property:** and there is no check of allowance/approval from the address owner	FNK, FCNE, FCE, FCNCE, FPNC	VC

Method

Step 3: Static analysis-based confirmation

- LLM used to find related variables for static vulnerability checking

An Example Prompt for GPT Recognition

System: (same as in Figure 4, omitted here for brevity.)

In this function, which variable or function holds the total supply/liquidity AND is used by the conditional branch to determine the supply/liquidity is 0? Please answer in a section starts with "VariableB:".
In this function, which variable or function holds the value of the deposit/mint/add amount? Please answer in a section starts with "VariableC:".
Please answer in the following json format: {"VariableA":{"Variable name":"Description"}, "VariableB":{"Variable name":"Description"}, "VariableC":{"Variable name":"Description"}}
[%CODE%]

Method

Step 3: Static analysis-based confirmation

Ask GPT model for related variables/expressions

- Map variable to vulnerability models

Validate the variables/expressions

- Validate the name
- Validate the description

Apply static check rules

- Dataflow
- Value Comparation
- Execution Order
- Function Call Arguments

Evaluation
Setup & Research Questions

NANYANG TECHNOLOGICAL UNIVERSITY

- Model selection
  - GPT-3.5 Turbo
- Dataset
  - Top 200 contracts from 6 chains: 303 projects, 0 logic vulnerability
  - Web3Bugs: 72 projects, 48 logic vulnerabilities
  - DefiHacks: 13 projects, 14 logic vulnerabilities
- Research Questions:
  - RQ1 & 2: How effective and precise is GPTScan?
  - RQ3: How effective is the static analysis-based confirmation?
  - RQ4: What’s the speed and financial cost of GPTScan?
  - RQ5: Could GPTScan find new vulnerabilities?

Evaluation

RQ1 & 2: Effectiveness and precision

- FP Rate:
  - Top 200: 4.39%
- Precision:
  - Web3Bugs: 57.14%
  - DefiHacks: 90.91%
- Recall:
  - Web3Bugs: 83.33%
  - DefiHacks: 71.43%

Dataset Name	TP	TN	FP	FN	Sum
Top200	0	283	13	0	296
Web3Bugs	40	154	30	8	232
DefiHacks	10	19	1	4	34

Evaluation

RQ1 & 2: Effectiveness and precision

- Baselines:
  - Slither:
    - Supported Types: Unauthorized Transfer (unchecked-transfer, arbitrary-send-eth, arbitrary-send-erc20)
    - 146 FPs, and 0 TPs on Web3Bugs
  - MetaScan:
    - Supported Types: Price Manipulation
    - Recall of 58.33% and precision of 100%

Dataset Name	TP	TN	FP	FN	Sum
Top200	0	283	13	0	296
Web3Bugs	40	154	30	8	232
DefiHacks	10	19	1	4	34

Evaluation

RQ3: Effectiveness of static confirmation

- Reduced nearly 2/3 FPs
- Caused only 1 FN

Vulnerability Type	Before	After
Approval Not Cleared	34	12
Risky First Deposit	100	21
Price Manipulation by AMM	187	114
Price Manipulation by Buying Tokens	8	8
Vote Manipulation by Flashloan	2	0
Front Running	6	4
Wrong Interest Rate Order	150	11
Wrong Checkpoint Order	49	1
Slippage	99	42
Unauthorized Transfer	12	8
Total	**647**	**221**

Evaluation

RQ4: Time and financial cost

- 14.39 seconds per thousand lines of code
- 0.01 USD per thousand lines of code

Dataset	KL*	T**	C***	T/KL	C/KL
Top200	134.32	1,437.37	0.7507	10.70	0.005589
Web3Bugs	319.88	4,980.57	3.9682	15.57	0.018658
DefiHacks	17.82	375.41	0.2727	21.06	0.015303
Overall	**472.02**	**6,793.35**	**4.9984**	**14.39**	**0.010589**

* KL for KLoC; ** T for Time; *** C for Financial Cost.

Evaluation

RQ5: Newly detected vulnerabilities

- Found 3 new vulnerabilities
  - 1 case of front running
  - 1 case of price manipulation
  - 1 case of risky first depositor

```
function deposit(uint _amount) external {
    ...
    uint _pool = balance();
    uint _totalSupply = totalSupply();
    if (_totalSupply == 0 && _pool > 0) { // trading fee accumulated while
        there were no IF LPs
        vusd.safeTransfer(governance, _pool);
        _pool = 0;
    }
    uint shares = 0;
    if (_pool == 0) {
        shares = _amount;
    } else {
        shares = _amount * _totalSupply / _pool;
    }
    ...
}
```

```
function pendingRewards(uint256 _pid, address _user) external view returns
    (uint256) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];
    uint256 accRewardsPerShare = pool.accRewardsPerShare;
    uint256 lpSupply = pool.lpToken.balanceOf(address(this));
    if (block.number > pool.lastRewardBlock && lpSupply != 0) {
        uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.
            number);
        uint256 rewardsAccum = multiplier.mul(rewardsPerBlock).mul(pool.
            allocPoint).div(totalAllocPoint);
        accRewardsPerShare = accRewardsPerShare.add(rewardsAccum.mul(1e12).
            div(lpSupply));
    }
    return user.amount.mul(accRewardsPerShare).div(1e12).sub(user.
        rewardDebt);
}
```

```
/// @notice The lp tokens that the user contributes need to have been
    transferred previously, using a batchable router.
function mint(address to)
    public
    beforeMaturity
    returns (uint256 minted)
{
    uint256 deposit = pool.balanceOf(address(this)) - cached;
    minted = _totalSupply * deposit / cached;
    cached += deposit;
    _mint(to, minted);
}
```

Summary

1. GPTScan is the **first tool** for logic vulnerability detection on smart contracts
2. GPTScan combined static program analysis with LLMs for both semantic understanding and precision
3. GPTScan is more effective than traditional tools on logic bugs
4. GPTScan is cheap and fast
5. GPTScan is extensive by adding more rules

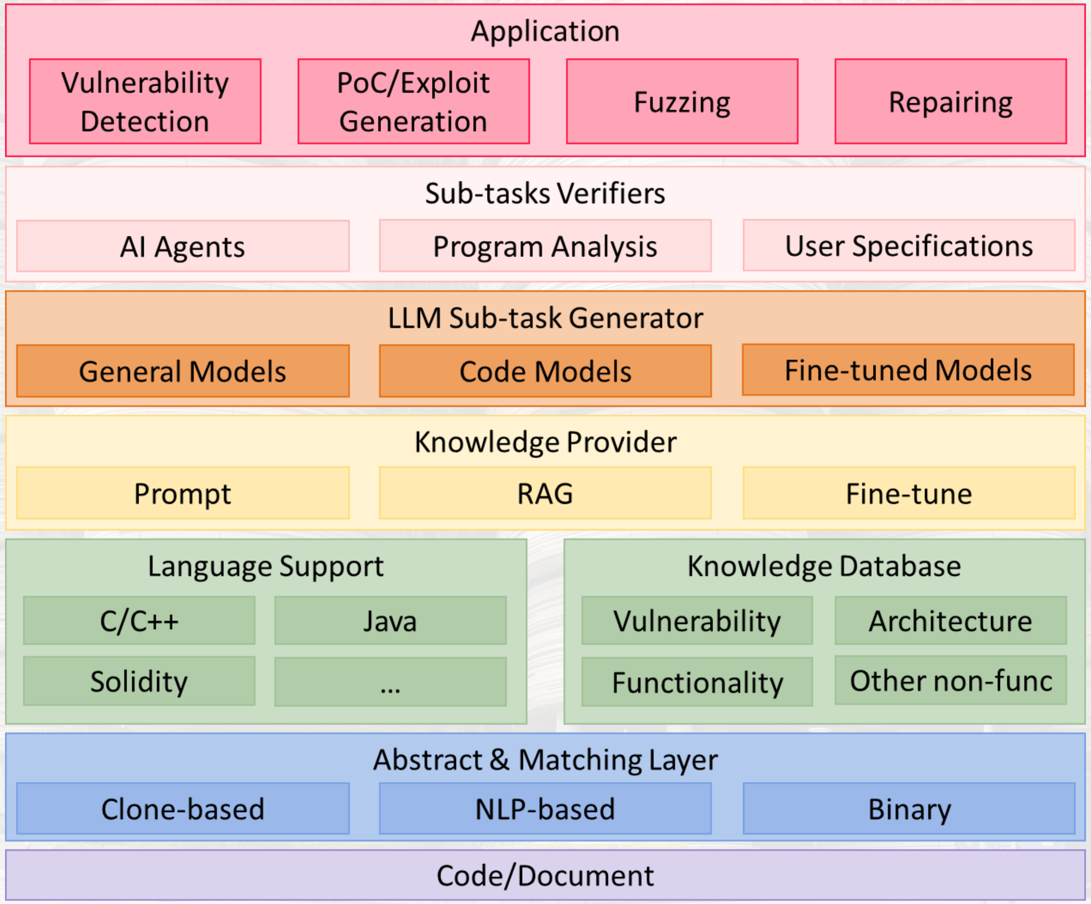
Limitations

NANYANG TECHNOLOGICAL UNIVERSITY

- Rule generation
  - Time-consuming for manually tuned rules
  - Low-accuracy for automatic generated (by LLM) rules
- Rule matching
  - Prompt based matching will not work when the number of rules increased
- These two problems are partially solved in our new preprint
  - [LLM4Vuln: A Unified Evaluation Framework for Decoupling and Enhancing LLMs' Vulnerability Reasoning]

Future AI4SE Framework

NANYANG TECHNOLOGICAL UNIVERSITY



Thanks & QA

Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, Yang Liu

Email: suny0056@e.ntu.edu.sg