GPTScan_But_Bigger

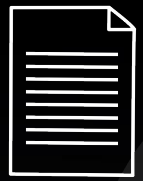
Based on: GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis

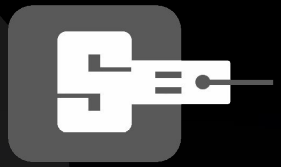
Conference: IEEE/ACM (2024) 46th International Conference on Software Engineering

Authors: Sun et al.

Presenters: Owen Joslin and William Joslin

Agenda





1. **GPTScan Overview**
2. **Background**
   + Key Components
   + Evaluation Framework
   + Technical Implementation
3. **Baselines**
4. **Limitations & Extensions**
5. **Repository Improvements**
   + Comments & Refactorization
   + Documentation
   + Shell Script Initialization
6. **Vulnerability Classification**
7. **Vulnerability Remediation**
8. **GPT Model Expansion**
9. **Conclusion**

1. GPTScan Overview

*GPTScan integrates **Generative Pre-training Transformer (GPT) models** with static analysis to detect **logic vulnerabilities in smart contracts**. Unlike conventional tools, it targets **business logic issues** rather than just structural vulnerabilities.*

*"Leveraging AI technology, such as **ChatGPT**, to detect logic vulnerabilities, encompassing a wider range of vulnerability types and adapting to various code variations. It enables detection even when the vulnerability names and code flow have been altered"*

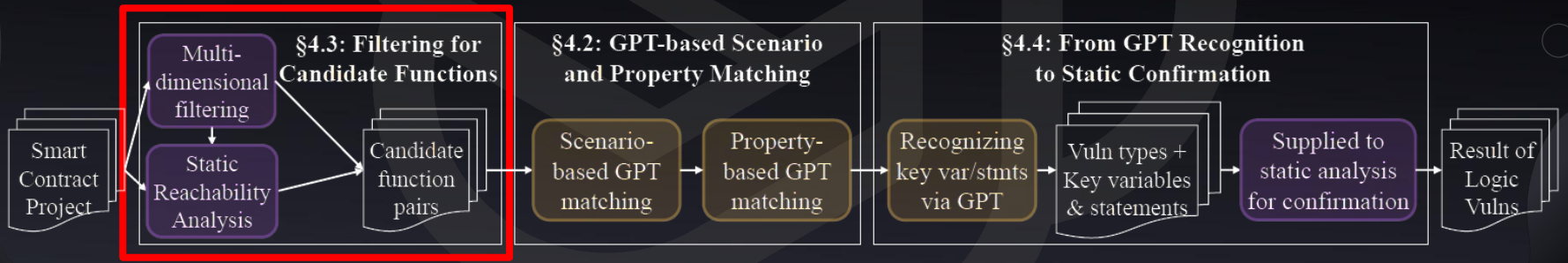
https://metatrust.io/

2. Background

Key Components of GPTScan:

★ **Multi-dimensional Filtering:** Candidate functions are pre-filtered:

- Non-Solidity files, libraries (OpenZeppelin), and irrelevant functions
- YAML-based rule system for targeted function filtering

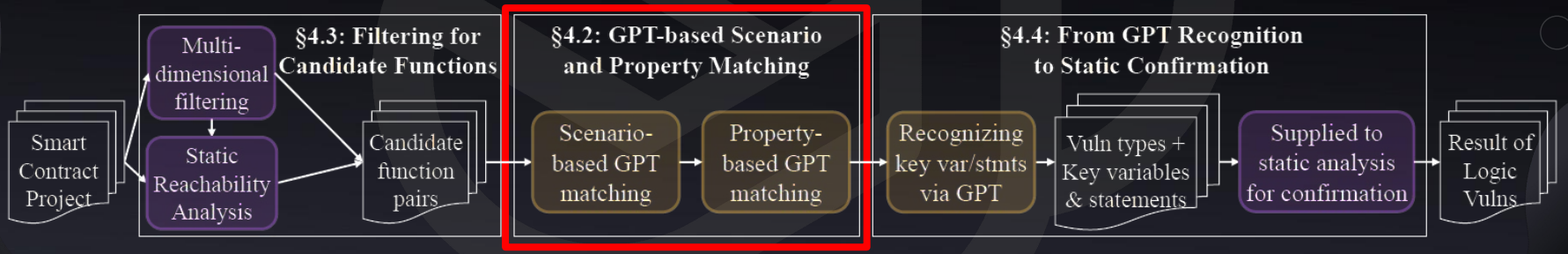


2. Background

Key Components of GPTScan:

★ **Scenario and Property Matching:**

- Vulns are split into 'scenarios' and 'properties'
- GPT then matches candidate functions against them

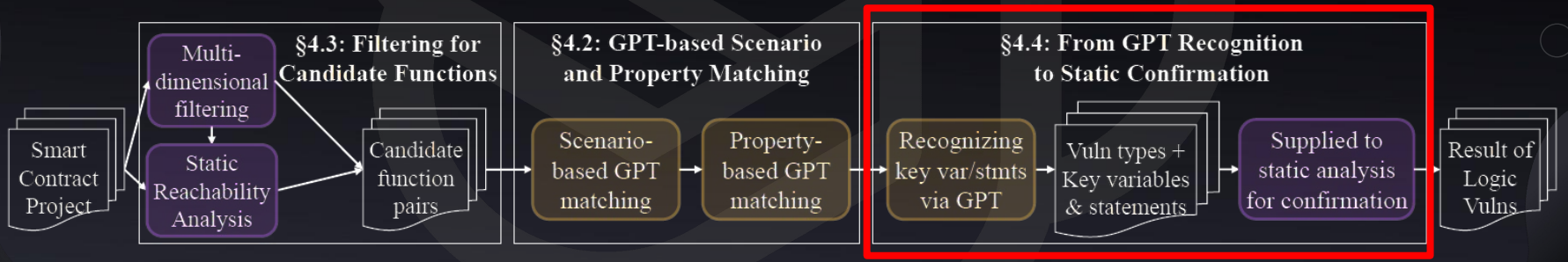


2. Background

Key Components of GPTScan:

★ **Static Confirmation:** GPT identifies variables and statements; then verified through static analysis to confirm vulnerabilities

- *Data Flow Tracing (DF)*
- *Value Comparison Check (VC)*
- *Order Check (OC)*
- *Function Argument Check (FA)*



3. Baselines

Evaluation Framework:

- ★ GPTScan was **tested on three datasets:**
  - ***Top200:*** Top 200 market capitalization smart contracts
  - ***Web3Bugs:*** Code4rena-audited projects with documented vulnerabilities
  - ***DefiHacks:*** Vulnerable contracts from past attack incidents

Confusion Matrix, Recall, Precision, Accuracy, F1

Technical Implementation:

- ★ Utilized gpt-3.5-turbo for cost efficiency
- ★ Static analysis leveraged ANTLR for code parsing
- ★ Crytic-compiler for dependency graph generation

3. Baselines

Accuracy & Effectiveness:

False Positive Rate:

★ For non-vulnerable contracts in Top200: **4.39%**

★ High precision for token contracts **(~90%)** and useable for larger projects **(~57%)**

Detection Metrics:

★ *Web3Bugs:* Recall of **83.33%**, F1 Score of **67.8%**

★ *DefiHacks:* Recall of **71.43%**, F1 Score of **80%**

Dataset Name	Projects P	Files F	F/P	LoC	Vuls
Top200	303	555	1.83	134,322	0
Web3Bugs	72	2,573	35.74	319,878	48
DefiHacks	13	29	2.23	17,824	14
Sum	388	3,157	8.14	472,024	62

3. Baselines

Cost & Performance: (gpt-3.5-turbo)

★ *Average time:* **14.39** seconds per 1,000 lines of Solidity code

★ *Average cost:* **$0.01** per 1,000 lines

Newly Discovered Vulnerabilities:

★ GPTScan identified **9** new vulnerabilities missed by human auditors in the Web3Bugs dataset

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

4. Limitations

Paper Identified:

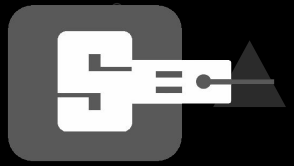
- ★ Lacks path-sensitive analysis
- ★ Pre-defined, while list filtering
- ★ Vulnerable to GPT’s inherent challenges like hallucination

4. Extensions

Proposed Extensions & Issues Identified:

- ★ GPT 3.5 and GPT 4 results only
  - gpt-4o, gpt-4o-mini, gpt-o1-preview, gpt-o1-mini (API access)
- ★ Messy, uncommented code-base; broken out-of-the-box; terrible documentation
  - Refactored and commented the codebase
  - Shell script-based initialization
  - Better documentation
- ★ Lack of vulnerability classification
  - Common Vulnerability Scoring System (CVSS) v2.0: Low, Medium, High
- ★ Lack of vulnerability remediation
  - Provide recommended code fix for identified vulnerabilities

5. Repository Improvements



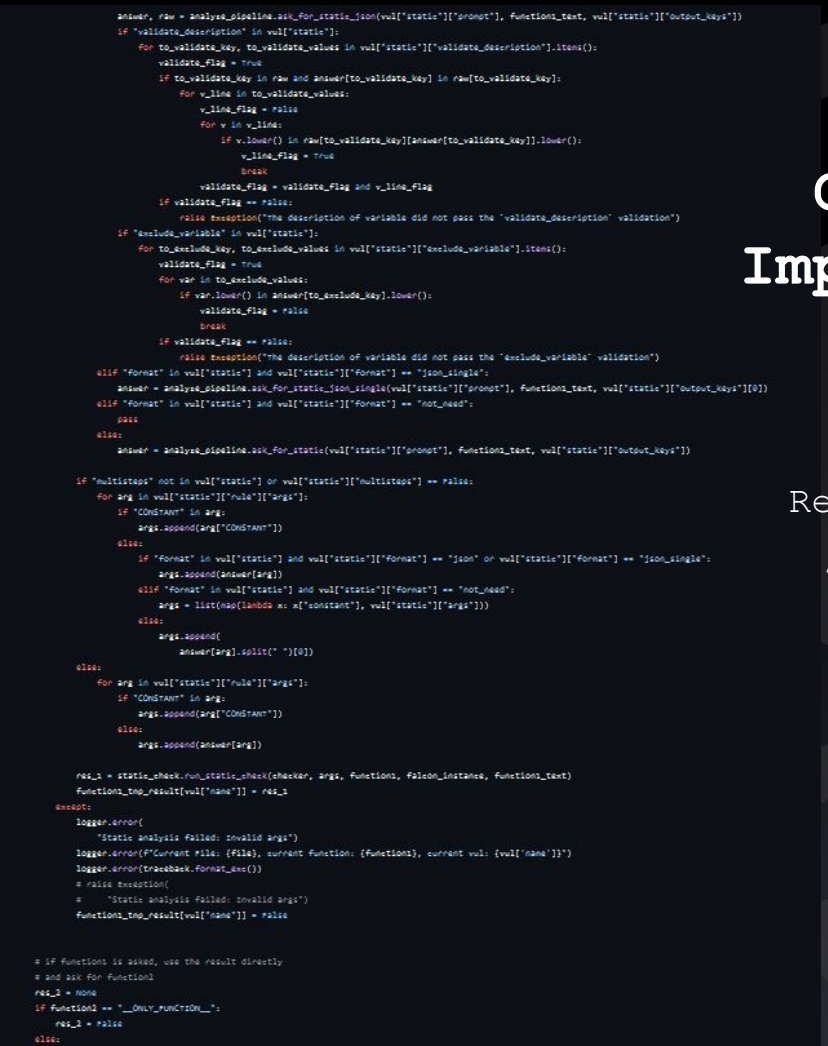
+ Comments & Refactorization
+ Documentation
+ Shell Script Initialization

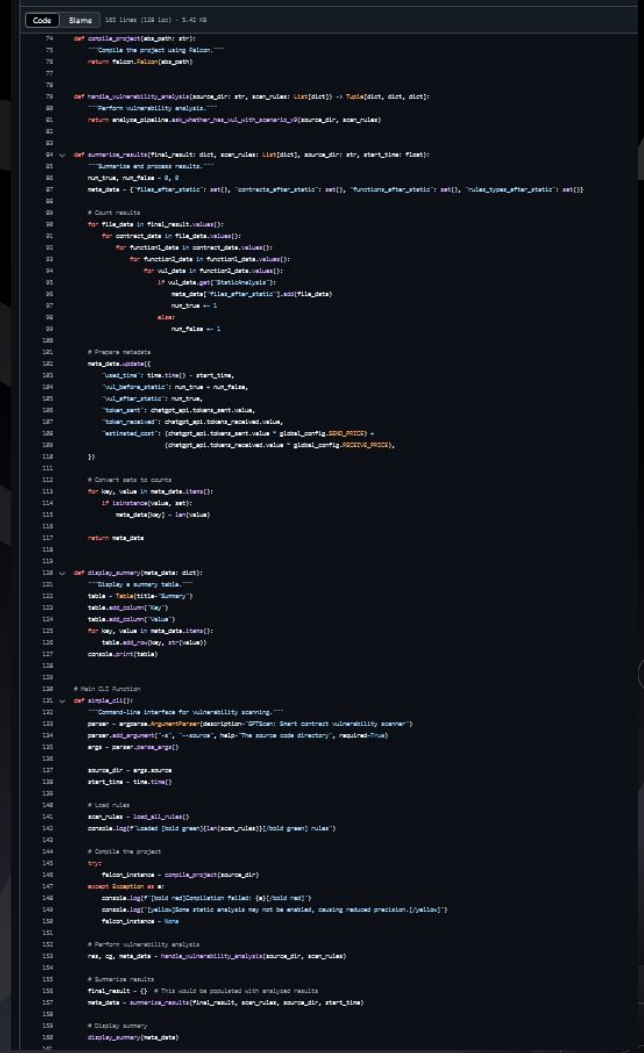
5. Codebase Improvements

Comments &

Refactorization

434->164 LOC





5. Codebase Improvements

Documentation

README AGPL-3.0 license

GPTScan

Description

Using ChatGPT for logic vulnerability detection.

How to Use

1. Install dependencies,

- Requires Python 3.10+
- Install Python dependencies: `pip install -r requirements.txt`

2. Run GPTScan

For example, if the source code is stored in the `/source` directory, run the command:

```
python3.10 main.py -s /sourcecode -o /sourcecode/output.json -k OPEN_AI_API_KEY_xxxxxxxxxxxx
```

3. Check the output

The output results are located at the location specified by the `-o` parameter, in the example above, it is located at `/sourcecode/output.json` .

Supported Project Types

Currently supported project types include:

- Single file, i.e., a single `.sol` file
- Multi-file, i.e., a directory with multiple `.sol` files, without any other external dependencies
- Common framework projects, such as Truffle, Hardhat, Brownie, etc.

Tested frameworks include:

- Hardhad
- Truffle
- Brownie

Note that this project does not include the compilation environment, such as Node.js, which needs to be installed separately.

Dataset

Dataset used to evaluate GPTScan in the paper, are the following:

1. Web3Bugs: https://github.com/MetaTrustLabs/GPTScan-Web3Bugs
2. DefiHacks: https://github.com/MetaTrustLabs/GPTScan-DefiHacks
3. Top200: https://github.com/MetaTrustLabs/GPTScan-Top200

How to Cite this project

```
@inproceedings{sun2024gptscan,
    author = {Sun, Yuqiang and Wu, Daoyuan and Xue, Yue and Liu, Han and Wang, Haijun and Xu, Zhengzi and Xie, Xiaofei and Liu, Yang},
    title = {{GPTScan}: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis},
    year = {2024},
```

README AGPL-3.0 license

GPTScan_Bigger_Model

Description

Using ChatGPT for logic vulnerability detection.

How to Use

1. Install dependencies,

- Usable only in UNIX environment (Tested in Ubuntu 24.0.1 LTS)
- Requires Python 3.10+ `apt-get install python3`
- Requires Java 11+ `apt-get install default-jre`
- Requires Node.js/nvm 0.40+ https://github.com/nvm-sh/nvm?tab=readme-ov-file#installing-and-updating
- Requires solc 0.8+ `add-apt-repository ppa:ethereum/Ethereum & apt update & apt-get install solc`
- Install Python dependencies: `pip install -r requirements.txt`

2. GPTScan Usage

Update `src/config.py` with the:

- GPT_API_KEY
- GPT_Model_Version

Move Solidity files to a usable directory following the directory structure provided by MetaTrustLabs/GPTScan-Top200 where each .sol file is in its own folder within the SOURCECODE_DIRECTORY. i.e /data/0x000000/test.sol where /data is SOURCECODE_DIRECTORY. NOTE: First run may be slow due to hardhat configuration and Solidity compilation.

```
./run_gptscan.sh /SOURCECODE_DIRECTORY
```

3. Check the output

The output results are located in each file's directory:

- Analysis output: `/data/0x000000/gptscan_output.json`
- Metadata output: `/data/0x000000/gptscan_output.metadata.json`
- GPTScan stdout: `/data/0x000000/gptscan_results.md` .

Supported Project Types

Currently supported project types include:

- Single file, i.e., a single `.sol` file
- Multi-file, i.e., a directory with multiple `.sol` files, without any other external dependencies
- Common framework projects, such as Truffle, Hardhat, Brownie, etc.

Tested frameworks include:

- Hardhat
- Truffle
- Brownie

Dataset

Dataset used to evaluate GPTScan in the paper, are the following:

1. Web3Bugs: https://github.com/MetaTrustLabs/GPTScan-Web3Bugs
2. DefiHacks: https://github.com/MetaTrustLabs/GPTScan-DefiHacks
3. Top200: https://github.com/MetaTrustLabs/GPTScan-Top200

5. Codebase Improvements

Shell Script Initialization

src > $ run_gptscan.sh

```bash
#!/bin/bash

# Check if a directory parameter was provided
if [[ -z "$1" ]]; then
    echo "Usage: $0 <test_directory>"
    exit 1
fi

# Use the provided directory parameter
test_directory="$1"
main_script_path="main.py"

# Run setup script with the provided directory
./setup_contracts.sh "$test_directory"

# Iterate over all directories in the test directory
for dir_path in "$test_directory"/*; do
    # Ensure it's a directory
    if [[ -d "$dir_path" ]]; then
        # Define the output file path
        output_file="$dir_path/gptscan_results.md"

        # Construct and execute the command, redirecting output to the file
        echo "Running command: python3 $main_script_path -s $dir_path"
        python3 "$main_script_path" -s "$dir_path" | tee "$output_file"

        # Check if the command failed
        if [[ $? -ne 0 ]]; then
            echo "Error while running command for directory: $dir_path" | tee
            -a "$output_file"
        fi
    fi
done
```

```
(.venv) owen@UbuntuVM:~/Documents/GitHub/GPTScan-Bigger-Model/src$ ./run_gptscan.sh /home/owen/Documents/GitHub/GPTScan-Bigger-Model/eval_data
```

5. Codebase Improvements

Shell Script Initialization

run_gptscan.sh

src > $ run_gptscan.sh

```
1  #!/bin/bash
2
3  # Check if a directory parameter was provided
4  if [[ -z "$1" ]]; then
5      echo "Usage: $0 <test_directory>"
6      exit 1
7  fi
8
9  # Use the provided directory parameter
10 test_directory="$1"
11 main_script_path="main.py"
12
13 # Run setup script with the provided directory
14 ./setup_contracts.sh "$test_directory"
15
16 # Iterate over all directories in the test directory
17 for dir_path in "$test_directory"/*; do
18     # Ensure it's a directory
19     if [[ -d "$dir_path" ]]; then
20         # Define the output file path
21         output_file="$dir_path/gptscan_results.md"
22
23         # Construct and execute the command, redirecting output to the file
24         echo "Running command: python3 $main_script_path -s $dir_path"
25         python3 "$main_script_path" -s "$dir_path" | tee "$output_file"
26
27         # Check if the command failed
28         if [[ $? -ne 0 ]]; then
29             echo "Error while running command for directory: $dir_path" | tee
               -a "$output_file"
30         fi
31     fi
32 done
```

5. Codebase Improvements

Shell Script Initialization

setup_contracts.sh

src > $ setup_contracts.sh

```
process_directory() {
    # Initialize npm project
    if [ ! -f "package.json" ]; then
        echo "Initializing npm project in $DIR..."
        yes "" | npm init -y || { echo "npm init failed in $DIR"; return; }
    else
        echo "npm project already initialized in $DIR."
    fi

    # Install Hardhat locally
    echo "Installing Hardhat in $DIR..."
    npm install hardhat --save-dev || { echo "Failed to install Hardhat in $DIR"; return; }

    # Initialize Hardhat with default settings
    if [ ! -f "hardhat.config.js" ]; then
        echo "Initializing Hardhat project in $DIR..."
        yes "" | npx hardhat || { echo "Failed to initialize Hardhat in $DIR"; return; }

        # Delete the default Lock.sol file
        echo "Removing Lock.sol from contracts directory in $DIR..."
        rm -f contracts/Lock.sol || echo "No Lock.sol file found to delete."
    else
        echo "Hardhat already initialized in $DIR."
    fi
```

5. Codebase Improvements

Shell Script Initialization

setup_contracts.sh

```
# Move all .sol files in the current directory to the contracts directory
echo "Moving .sol files to contracts/ in $DIR..."
mkdir -p contracts
find . -maxdepth 1 -type f -name "*.sol" -exec mv {} contracts/ \;
```

```
# Move all .sol files in the current directory to the contracts directory
echo "Moving .sol files to contracts/ in $DIR..."
mkdir -p contracts
find . -maxdepth 1 -type f -name "*.sol" -exec mv {} contracts/ \;

# Extract Solidity versions from .sol files in contracts/ and update hardhat.config.js
echo "Extracting Solidity versions from .sol files in $DIR..."
SOLIDITY_VERSIONS=()
for SOL_FILE in contracts/*.sol; do
    if [ -f "$SOL_FILE" ]; then
        VERSION_LINE=$(grep -E "^pragma solidity" "$SOL_FILE" | head -n 1)
        if echo "$VERSION_LINE" | grep -qE "^pragma solidity [^;]+;"; then
            VERSION=$(echo "$VERSION_LINE" | sed -E 's/^pragma solidity[^0-9]*([0-9]+\.[0-9]+\.[0-9]+).*/\1/')
            SOLIDITY_VERSIONS+=("$VERSION")
        fi
    fi
done

# Remove duplicates from SOLIDITY_VERSIONS
UNIQUE_VERSIONS=($(echo "${SOLIDITY_VERSIONS[@]}" | tr ' ' '\n' | sort -u | tr '\n' ' '))

# Update hardhat.config.js with the extracted versions
if [ ${#UNIQUE_VERSIONS[@]} -gt 0 ]; then
    echo "Adding Solidity versions to hardhat.config.js in $DIR..."
    COMPILERS_STRING=$(printf '{ version: "%s" },\n            ' "${UNIQUE_VERSIONS[@]}")
    COMPILERS_STRING=${COMPILERS_STRING%, } # Remove trailing comma

    HARDHAT_CONFIG="require(\"@nomicfoundation/hardhat-toolbox\");

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
    solidity: {
        compilers: [
            $COMPILERS_STRING
        ]
    }
};"

    echo "$HARDHAT_CONFIG" > hardhat.config.js
else
    echo "No Solidity versions found in .sol files in $DIR."
fi
```

```
# Compile the smart contracts
echo "Compiling contracts in $DIR..."
npx hardhat compile || echo "Compilation failed in $DIR."
```

5. Codebase Improvements

Shell Script Initialization

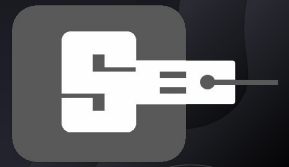
run_gptscan.sh

src > $ run_gptscan.sh

```
#!/bin/bash

# Check if a directory parameter was provided
if [[ -z "$1" ]]; then
    echo "Usage: $0 <test_directory>"
    exit 1
fi

# Use the provided directory parameter
test_directory="$1"
main_script_path="main.py"

# Run setup script with the provided directory
./setup_contracts.sh "$test_directory"

# Iterate over all directories in the test directory
for dir_path in "$test_directory"/*; do
    # Ensure it's a directory
    if [[ -d "$dir_path" ]]; then
        # Define the output file path
        output_file="$dir_path/gptscan_results.md"

        # Construct and execute the command, redirecting output to the file
        echo "Running command: python3 $main_script_path -s $dir_path"
        python3 "$main_script_path" -s "$dir_path" | tee "$output_file"

        # Check if the command failed
        if [[ $? -ne 0 ]]; then
            echo "Error while running command for directory: $dir_path" | tee
            -a "$output_file"
        fi
    fi
done
```

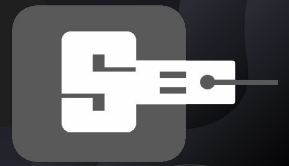
6. Vulnerability Classification

```
function rateVulnerability(codeSnippet)
    # Uses ChatGPT to analyze the code five times and aggregates the results
    # Provides average CVSS score, severity, and a consensus classification and recommendation

function parseResponse(response)
    # Parses the response received from ChatGPT
    # Extracts CVSS score, severity, CVE-like classification, and recommendation from the
response

function getSeverityFromValue(value, severityMapping)
    # Converts a numerical severity value back to the corresponding severity label
    # Uses closest-match approach to determine the correct severity level (e.g., Low, Medium,
etc.)

function mostCommon(elements)
    # Finds the most frequently occurring element in a list
    # Determines the most common classification or recommendation

function main()
    # Provides a sample code snippet for evaluation
    # Calls rateVulnerability() and prints the resulting vulnerability report
```

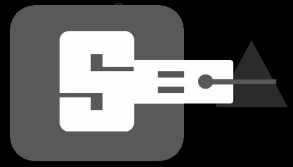


6. Vulnerability Classification

```
def rateVulnerability(codeSnippet):
    # Create a prompt to instruct ChatGPT
    prompt = """
    You are a cybersecurity expert. Evaluate the following code snippet for vulnerabilities.

    Code Snippet:
    {codeSnippet}

    Assess the vulnerabilities using CVSS (Common Vulnerability Scoring System) and CVE classifications.
    Provide the following specific values:
    - CVSS Score: (range 0.0 - 10.0)
    - CVSS Severity: (None, Low, Medium, High, Critical)
    - CVE-like Classification: (Provide a brief description of the vulnerability, if applicable)
    - Recommendation: (Short recommendation to mitigate the identified vulnerability)

    Return values in the following format:
    {
      "cvss_score": <score>,
      "cvss_severity": "<severity>",
      "cve_classification": "<description>",
      "recommendation": "<recommendation>"
    }
    """
```

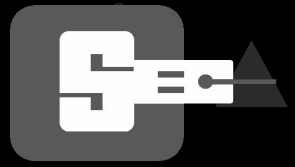


7. Vulnerability Remediation



```
function remediateVulnerability(codeSnippet)
    # Takes a vulnerable code snippet and provides secure alternatives along with best practices.

function suggestBestPractices(codeSnippet)
    # Analyzes the provided code and lists best practices to mitigate identified vulnerabilities.

function main(vulnerableCode)
    # Takes a vulnerable code snippet as input
    # Calls remediateVulnerability to generate secure alternatives
    # Calls suggestBestPractices to retrieve best practices
    # Returns the secure code snippets and best practices in JSON format

    secureCode = remediateVulnerability(vulnerableCode)
    bestPractices = suggestBestPractices(vulnerableCode)

    return {
        "secureCodeExamples": secureCode,
        "bestPractices": bestPractices
    }
```

8. GPT Model Expansion



Adoption for new models:

src > config.py.example

```
1  import logging
2  import datetime
3
4  # OpenAI API configuration
5  OPENAI_API_KEY = "YOUR_OPENAI_API_KEY"  # Replace with your actual OpenAI API k
6  MODEL = "gpt-4"  # Specify the model parameter for GPT's API (e.g., gpt-3.5-turl
7
8  # Optional configuration for other services or tokens
9  GITHUB_TOKEN = "NOT NEEDED"
```

MODEL	CONTEXT WINDOW	MAX OUTPUT TOKENS	TRAINING DATA
o1-preview Points to the most recent snapshot of the o1 model: `o1-preview-2024-09-12`	128,000 tokens	32,768 tokens	Up to Oct 2023
o1-preview-2024-09-12 Latest o1 model snapshot	128,000 tokens	32,768 tokens	Up to Oct 2023
o1-mini Points to the most recent o1-mini snapshot: `o1-mini-2024-09-12`	128,000 tokens	65,536 tokens	Up to Oct 2023
o1-mini-2024-09-12 Latest o1-mini model snapshot	128,000 tokens	65,536 tokens	Up to Oct 2023

MODEL	CONTEXT WINDOW	MAX OUTPUT TOKENS	TRAINING DATA
gpt-4o Our high-intelligence flagship model for complex, multi-step tasks. GPT-4o is cheaper and faster than GPT-4 Turbo. Currently points to `gpt-4o-2024-08-06`.	128,000 tokens	16,384 tokens	Up to Oct 2023
gpt-4o-2024-11-20 Latest `gpt-4o` snapshot from November 20th, 2024.	128,000 tokens	16,384 tokens	Up to Oct 2023
gpt-4o-2024-08-06 First snapshot that supports Structured Outputs. `gpt-4o` currently points to this version.	128,000 tokens	16,384 tokens	Up to Oct 2023
gpt-4o-2024-05-13 Original `gpt-4o` snapshot from May 13, 2024.	128,000 tokens	4,096 tokens	Up to Oct 2023
chatgpt-4o-latest The `chatgpt-4o-latest` model version continuously points to the version of GPT-4o used in ChatGPT, and is updated frequently, when there are significant changes.	128,000 tokens	16,384 tokens	Up to Oct 2023

9. Conclusion

Paper Identified Limitations:

★ Lacks path-sensitive analysis
★ Pre-defined, while list filtering
★ Vulnerable to GPT’s inherent challenges like hallucination

Vulnerability Classification

★ Larger vulnerability set
★ Context awareness

Vulnerability Remediation

★ GPT hallucinations

Model Expansion

★ Needs to cook a little more
★ o1-preview 15$ per 1mil tokens

Questions_?
_Demo?