

# GPTScan\_But\_Bigger

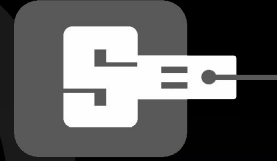
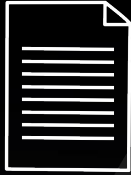
**Based on:** GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by  
Combining GPT with Program Analysis

**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"*

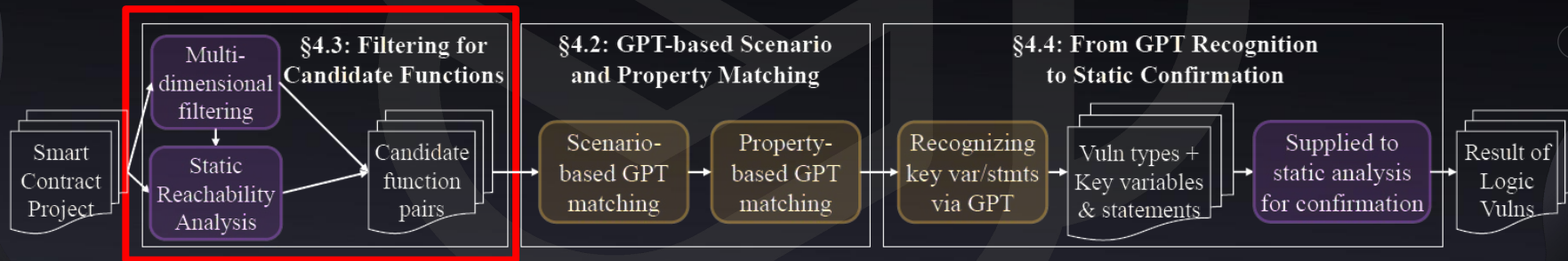
<https://metatrust.io/>

## 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



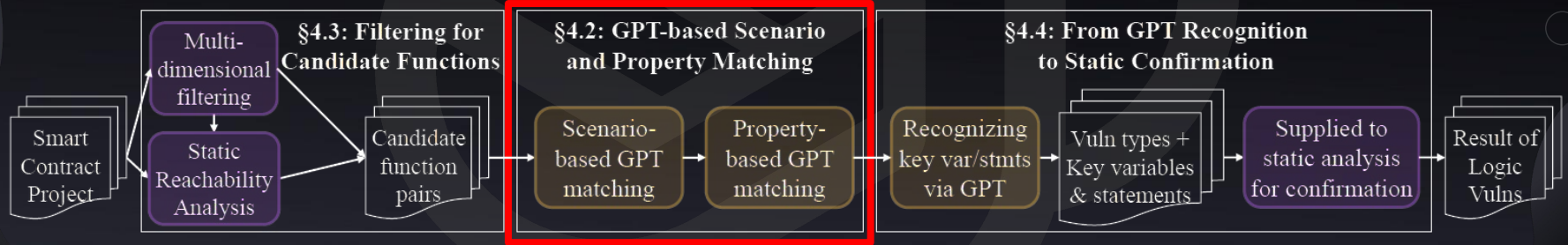
## 2. Background



**Key Components of GPTScan:**

★ **Scenario and Property Matching:**

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

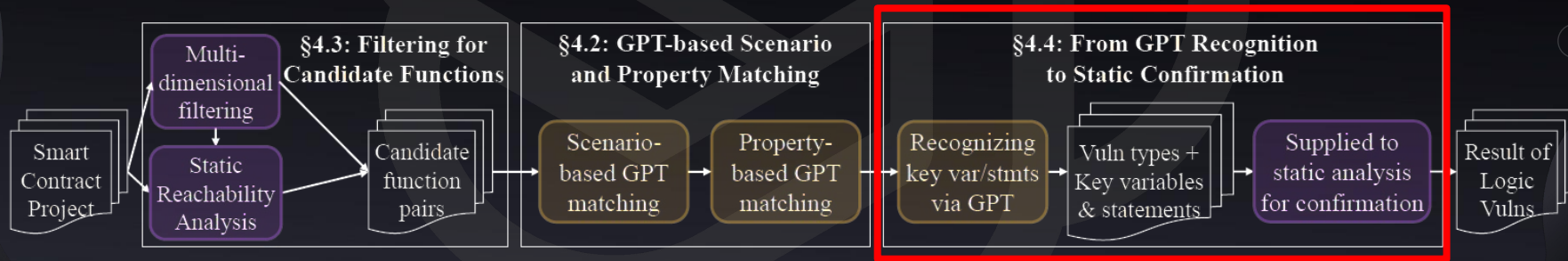


## 2. Background



### 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
<b>Sum</b>	<b>388</b>	<b>3,157</b>	<b>8.14</b>	<b>472,024</b>	<b>62</b>





### 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

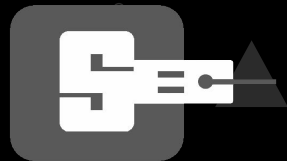
Dataset Name	TP	TN	FP	FN	Sum
Top200	0	283	13	0	296
Web3Bugs	40	154	30	8	232
DefiHacks	10	19	1	4	34



## 4. Limitations

### Paper Identified:

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



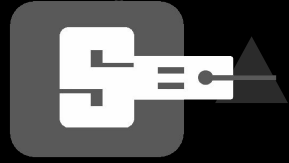
## 4. Extensions

### Proposed Extensions & Issues Identified:

- ★ GPT 3.5 and GPT 4 results only
  - gpt-4o, gpt-4o-mini, gpt-ol-preview, gpt-ol-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



```
answer, raw = analyze_pipeline.ask_for_static(ison(vul["static"]["prompt"], functions_text, vul["static"]["outout_keys"])
if "validate_description" in vul["static"]:
    for to_validate_key, to_validate_values in vul["static"]["validate_description"].items():
        validate_flag = True
        if to_validate_key in raw and answer[to_validate_key] in raw[to_validate_key]:
            for v_line in to_validate_values:
                v_line_flag = False
                for v in v_line:
                    if v.lower() in raw[to_validate_key][answer[to_validate_key]].lower():
                        v_line_flag = True
                        break
                validate_flag = validate_flag and v_line_flag
            if validate_flag == False:
                raise Exception("The description of variable did not pass the 'validate_description' validation")
        if "exclude_variable" in vul["static"]:
            for to_exclude_key, to_exclude_values in vul["static"]["exclude_variable"].items():
                validate_flag = True
                for var in to_exclude_values:
                    if var.lower() in answer[to_exclude_key].lower():
                        validate_flag = False
                        break
            if validate_flag == False:
                raise Exception("The description of variable did not pass the 'exclude_variable' validation")
        elif "format" in vul["static"] and vul["static"]["format"] == "json_single":
            answer = analyze_pipeline.ask_for_static_ison_single(vul["static"]["prompt"], functions_text, vul["static"]["outout_keys"])[0]
        elif "format" in vul["static"] and vul["static"]["format"] == "not_need":
            pass
        else:
            answer = analyze_pipeline.ask_for_static(vul["static"]["prompt"], functions_text, vul["static"]["outout_keys"])

if "multisteps" not in vul["static"] or vul["static"]["multisteps"] == False:
    for arg in vul["static"]["rule"]["args"]:
        if "CONSTANT" in arg:
            args.append(arg["CONSTANT"])
        else:
            if "format" in vul["static"] and vul["static"]["format"] == "json" or vul["static"]["format"] == "json_single":
                args.append(answer[arg])
            elif "format" in vul["static"] and vul["static"]["format"] == "not_need":
                args = list(map(lambda x: x["constant"], vul["static"]["args"]))
            else:
                args.append(
                    answer[arg].split(" ")[0])
    else:
        for arg in vul["static"]["rule"]["args"]:
            if "CONSTANT" in arg:
                args.append(arg["CONSTANT"])
            else:
                args.append(answer[arg])

res_1 = static_check.run_static_check(checker, args, functions, falcon_instances, functions_text)
functions_top_result[vul["name"]] = res_1

except:
    logger.error(
        "Static Analysis Failed: Invalid args")
    logger.error(f"Current file: {file}, current function: {functions}, current vul: {vul['name']}")
    logger.error(traceback.format_exc())
    # raise Exception(
    #     "Static analysis failed: Invalid args")
    functions_top_result[vul["name"]] = False

# If Functions is asked, use the result directly
# and ask for function1
res_2 = None
if function1 == "ONLY_FUNCTION_1":
    res_2 = False
else:
    try:
        args = []
```

```
Code | Blame | 165 lines (120 loc) | 5-40 KB
74 def compile_project(data_dir: str):
75     """Compile the project using Falcon."""
76     return Falcon.compile(data_dir)
77
78
79 def handle_vulnerability_analysis(data_dir: str, source_dir: list[str]) -> tuple[dict, dict, dict]:
80     """Perform vulnerability analysis."""
81     return analyze_pipeline.ask_function_name_vuln_with_comments(data_dir, source_dir)
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

# 5. Codebase Improvements

## Documentation

### 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 OPBIL_AI_API_KEY_xxxxxxxxxxxxxxxxxx
```

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:

- Hardhat
- 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, Yuxiang and Wu, Daoyuan and Xue, Yun and Liu, Han and Wang, Haijun and Xu, Zhengxi and Xie, Xiao  
  title = {(GPTScan): Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis},  
  year = {2024},
```

### GPTScan\_Bigger\_Model

#### Description

Using ChatGPT for logic vulnerability detection.

#### How to Use

1. Install dependencies.

- Usable only in UNIX environment (Tested in Ubuntu 24.01 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
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: $dir_path" | tee
30             -a "$output_file"
31         fi
32     fi
33 done
```



## 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: $dir_path" | tee
30             -a "$output_file"
31         fi
32     fi
33 done
```



## 5. Codebase Improvements

Shell Script

Initialization

`setup_contracts.sh`



```
src > $ setup_contracts.sh
17  process_directory() {
24      # Initialize npm project
25      if [ ! -f "package.json" ]; then
26          echo "Initializing npm project in $DIR..."
27          yes "" | npm init -y || { echo "npm init failed in $DIR"; return; }
28      else
29          echo "npm project already initialized in $DIR."
30      fi
31
32      # Install Hardhat locally
33      echo "Installing Hardhat in $DIR..."
34      npm install hardhat --save-dev || { echo "Failed to install Hardhat in $DIR"; return; }
35
36      # Initialize Hardhat with default settings
37      if [ ! -f "hardhat.config.js" ]; then
38          echo "Initializing Hardhat project in $DIR..."
39          yes "" | npx hardhat || { echo "Failed to initialize Hardhat in $DIR"; return; }
40
41          # Delete the default Lock.sol file
42          echo "Removing Lock.sol from contracts directory in $DIR..."
43          rm -f contracts/Lock.sol || echo "No Lock.sol file found to delete."
44      else
45          echo "Hardhat already initialized in $DIR."
46      fi
47  }
```

## 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[^\(]+\([0-9]+\.[0-9]+\.[0-9]+\).*/\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
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: $dir_path" | tee
30             -a "$output_file"
31         fi
32     fi
33 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
```

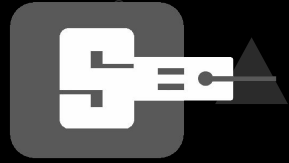


## 6. Vulnerability Classification

```
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 key
6  MODEL = "gpt-4" # Specify the model parameter for GPT's API (e.g., gpt-3.5-turbo)
7
8  # Optional configuration for other services or tokens
9  GITHUB_TOKEN = "NOT_NEEDED"
```

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

MODEL	CONTEXT WINDOW	MAX OUTPUT TOKENS	TRAINING DATA
<b>gpt-4o</b> 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
<b>gpt-4o-2024-11-20</b> Latest gpt-4o snapshot from November 20th, 2024.	128,000 tokens	16,384 tokens	Up to Oct 2023
<b>gpt-4o-2024-08-06</b> First snapshot that supports <b>Structured Outputs</b> . gpt-4o currently points to this version.	128,000 tokens	16,384 tokens	Up to Oct 2023
<b>gpt-4o-2024-05-13</b> Original gpt-4o snapshot from May 13, 2024.	128,000 tokens	4,096 tokens	Up to Oct 2023
<b>chatgpt-4o-latest</b> 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?\_