

Neuro-Symbolic Static Analysis for Reliable Software Systems

Chengpeng Wang
Postdoc, Purdue University

Joint works with

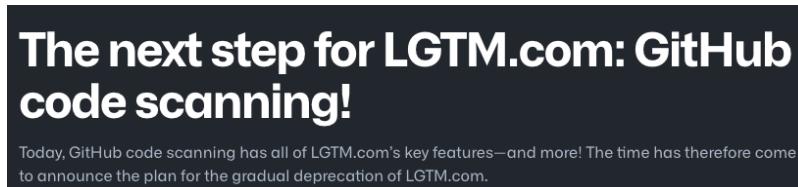
Jinyao Guo, Wuqi Zhang, Xiangzhe Xu,
Zian Su, and Xiangyu Zhang

Outline

- Static Bug Detection
- LLM-driven Data-flow Bug Detection
- Neuro-Symbolic Static Analysis

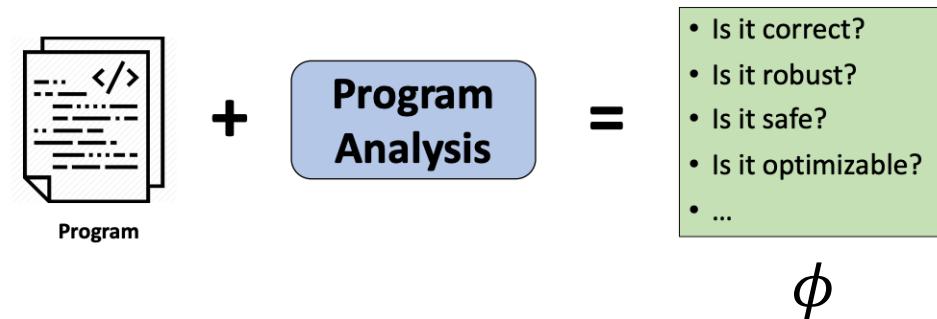
Programming in the LLM Era

- *Copy-and-paste* the code from intelligent search engine
- *Prompt-and-paste* the code from LLM bots
- *Comment-and-select* the code recommended by LLM-powered IDE



A Nature Shift of Programming

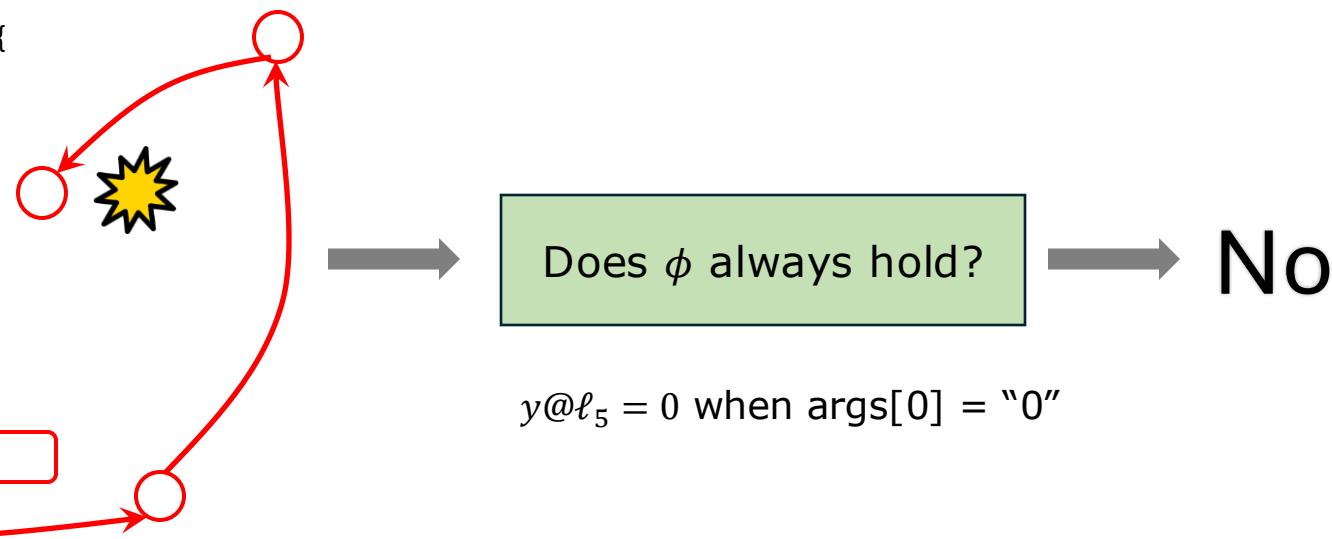
- Shift from writing code to validating correctness
 - Writing code is cheap
 - Validating correctness is expensive
 - Critical for software reliability
- Static analysis: Reason the program **statically** without execution
 - Determine whether a specific property ϕ holds for any inputs



Divide-by-Zero (DBZ) Bug Detection

- Target property ϕ : All the divisors are not equal to 0
- Rule-based symbolic analysis discovers a data-flow path
 - From the numeric literals 0, 0.0, and unconstrained inputs
 - To the divisors

```
1 public static void bar(int x, int y){  
2     if (x != 0)  
3         return (y * 1.0 / x);  
4     else  
5         return (x * 1.0 / y);  
6 }  
7 public static void main(){  
8     int a = 0;  
9     int b = parseInt("123");  
10    System.out.println(bar(b, a));  
11    String arg = args[0];  
12    int c = parseInt(arg);  
13    System.out.println(bar(a, c));  
14    c = b;  
15    System.out.println(bar(a, c));  
16 }
```



Data-flow Bugs

- Introduced by incorrect or unexpected propagation of data values
 - **Source:** Faulty values
 - **Sink:** The operands of dangerous operations
- Most of top 25 CWE (Common Weakness Enumeration) bug types are data-flow bugs
 - **Memory bugs:** Null pointer dereference, Memory leak, Use-after-free
 - **Taint-style Vulnerabilities:** SQL injection, Cross-site scripting
- Categorization
 - Source must not reach sink: Divide-by-Zero, Null pointer dereference, Taint-style Vulnerabilities
 - Source must reach sink: Memory leak



Mainstream Static Bug Detectors

- Existing effort: Make it more **precise**, more **efficient**, and more **scalable**
- Not widely deployed as expected [ICSE'13, ASE'16, FSE'24]
- Limitations
 - Compilation reliance
 - Customization obstacle



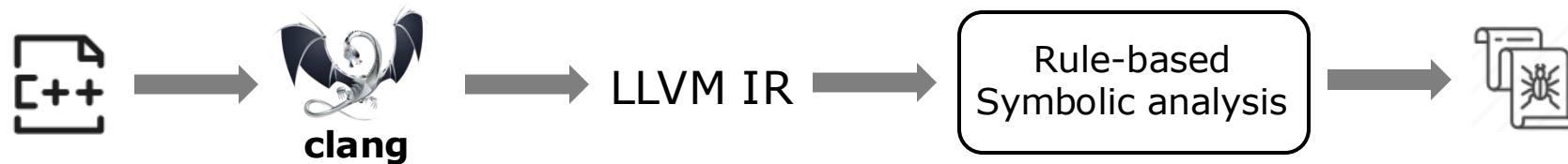
Brittany J., et al. Why Don't Software Developers Use Static Analysis Tools to Find Bugs? ICSE 2013

Maria Christakis and Christian Bird. What Developers Want and Need from Program Analysis: An Empirical Study. ASE 2016

Huaien Z., et al. Understanding and Detecting Annotation-Induced Faults of Static Analyzers? FSE 2024

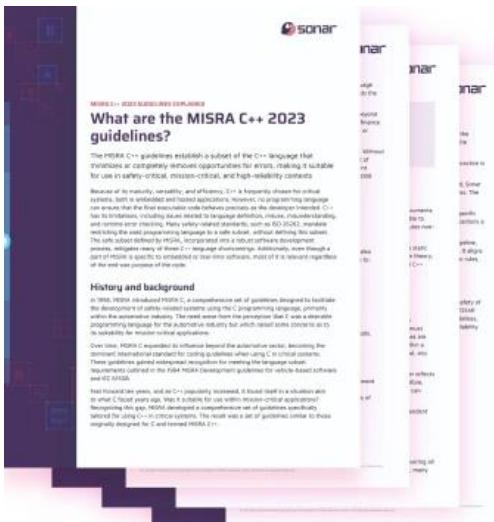
Limitation I: Compilation Reliance

- Existing static bug detectors mostly require intermediate representations (IRs) of programs generated by successful compilation
 - E.g., Facebook Infer, CodeQL, and SVF
 - Compatible with different language versions and convenience the implementation of analyzers
- Fail to discover security vulnerabilities in the uncompilable code
 - **Complex build environment:** Compiler version, program dependencies, etc
 - **Incomplete code:** Program under the development, code snippets generated by AI



Limitation II: Customization Obstacle

- Existing static bug detectors only target specific bug types and **can not support the user-friendly customization**
 - Require the **expert knowledge** on hacking the **compiler infrastructure**
 - Existing static bug detectors require **manually specified specifications**, such as library semantic specifications, to understand “dark code”.
 - Require **laborious manual effort**



Dynamic code execution should not be vulnerable to injection attacks

Vulnerability

NoSQL operations should not be vulnerable to injection attacks

Vulnerability

HTTP request redirections should not be open to forging attacks

Vulnerability

Deserialization should not be vulnerable to injection attacks

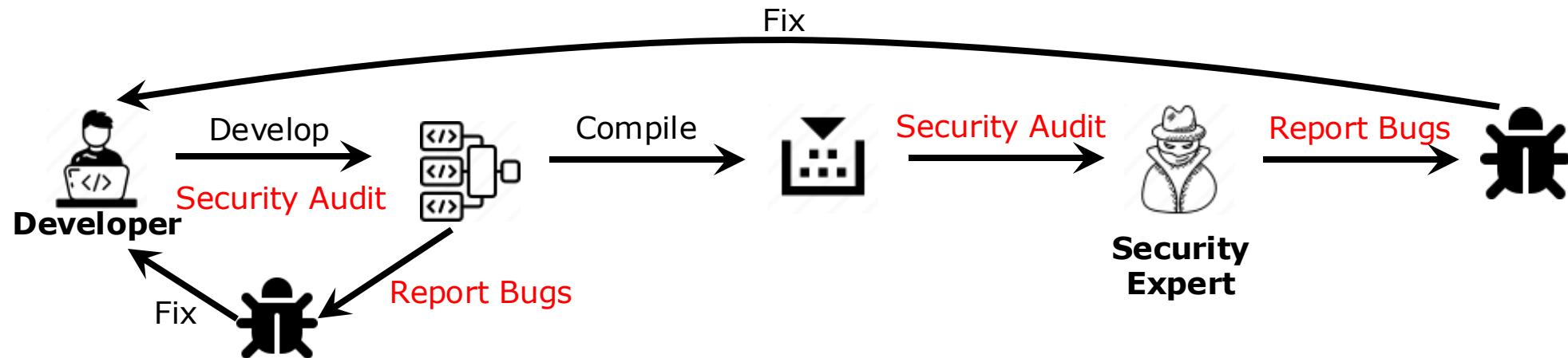
Vulnerability

```
public static void main(){
    int a = 0;
    Box container = Box();
    container.set(a);
    int c = container.get();
    int d = a / c;
}
```

set, get: impl available

Our Goal

- Build a ***compilation-free*** and ***customizable*** static analysis
 - No reliance on build/compilation
 - No reliance on expertise or intensive labor work
- **Leftward shift** of security auditing: Detect bugs actively during the development-time



Outline

- Static Bug Detection
- LLM-driven Data-flow Bug Detection
- Neuro-Symbolic Static Analysis

Reshaping Static Analysis with LLMs

CHAT

Please mimic the execution of the program and detect divide-by-zero bugs.

J demo.java:1-26

GitHub Copilot > Used 1 reference

Claude-3.5-Sonnet

I'll analyze the code execution step by step and identify potential divide-by-zero bugs.

Analysis

1. In main() :

- x = 1
- y = 10 / x → y = 10 (safe, x = 1)
- Create ArrayList and add: [1, 10, 0]
- y = x * 1 → y = 1
- z = x / y → z = 1 (safe, y = 1)
- z = list.get(2) → z = 0
- System.out.println(y / z) → DIVIDE BY ZERO ERROR! (y = 1, z = 0)
- Program would crash here, but if it continued:
- x = foo(y, z) with y = 1, z = 0
- Inside foo() : doesn't print since Math.abs(0) ≤ 1
- Returns z which is 0
- System.out.println(x / x) → DIVIDE BY ZERO ERROR! (x = 0)

Divide-by-Zero Bugs Found:

- Line 21: System.out.println(y / z) - when z = 0
- Line 24: System.out.println(x / x) - when x = 0

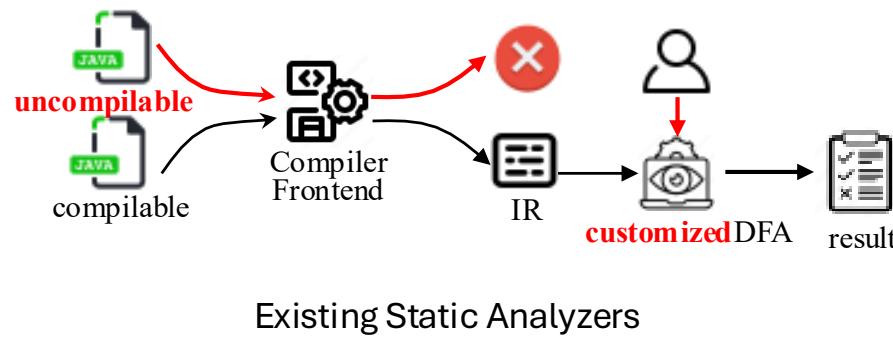
J demo.java 1 ×

Users > xiangqian > Downloads > J demo.java > ...

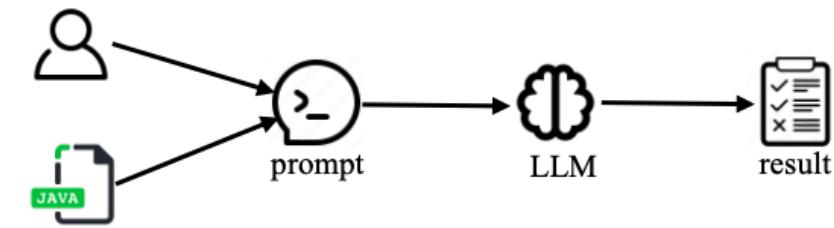
```
1 import java.util.ArrayList;
2
3 public class Demo {
4
5     public static int foo(int a, int b) {
6         if (Math.abs(b) > 1)
7             System.out.println(a / b);
8         return b;
9     }
10
11    public static void main(String[] args) {
12        int x = 1;
13        int y = 10 / x;
14        ArrayList<Integer> list = new ArrayList<>();
15        list.add(x);
16        list.add(y);
17        list.add(0);
18        y = x * 1;
19        int z = x / y;
20        z = list.get(2);
21        System.out.println(y / z);
22
23        x = foo(y, z);
24        System.out.println(x / x);
25    }
26 }
```

Our Vision

- LLMs for static analysis
 - Analyze code as specific kind of textual data
 - No need to build the code
 - Easy to customize the analysis
 - Analyze code via prompting
 - Serve as knowledge base of specifications
 - Pretrained upon huge data



Existing Static Analyzers



Our Proposal

LLMs are NOT Silver Bullets for Static Analysis

- Non-linear contexts exacerbate LLM hallucinations, causing many FPs/FNs
 - Examples of **non-linear program structures**: **CFG**, **PDG**, **Call Graph**
- Empirical study [arXiv 2024]
 - Select 100 buggy functions
 - Before and after fixes
 - Precision: ~30%
- Even worse in repo-level detection
 - **Even lower precision and recall** due to nonlinearity of call graph
 - Potential **huge overhead** in scanning the repository

Chen Y, et al. Dependency-Aware Code Naturalness, OOPSLA 2024.

Steenhoek B, et al. A Comprehensive Study of the Capabilities of Large Language Models for Vulnerability Detection, arXiv 2024.

Our Contribution

- Establish LLM-driven data-flow bug detection with **high precision, recall, and scalability**
 - **187 memory bugs** detected in high-profiled open-sourced projects since Feb 2025
 - 80 fixed bugs + 90 confirmed bugs
 - **Seven zero-days** in DARAPA AIxCC Ngnix Challenge
 - Invited talks in RSAC 2025 and GitHub CodeQL team

31	challenge-004-nginx	C	NPD
32	challenge-004-nginx	C	NPD
33	challenge-004-nginx	C	NPD
34	challenge-004-nginx	C	Memory Leak
35	challenge-004-nginx	C	Memory Leak
36	challenge-004-nginx	C	Memory Leak
37	challenge-004-nginx	C	Memory Leak



Advances in AI-Powered Code Security: Next-Level Bug Detection. GitHub talk, <https://www.youtube.com/watch?v=nOS56VC0FTQ>

LLM-driven Data-flow Bug Detection

- High Precision
 - LLMSAN: A sanitization technique that validates the LLM-generated bug reports
- High Recall
 - LLMDFA: A summary-based LLM-driven inter-procedural data-flow analyzer
- High Scalability
 - RepoAudit: An autonomous LLM-agent that explores the code repository for bug detection

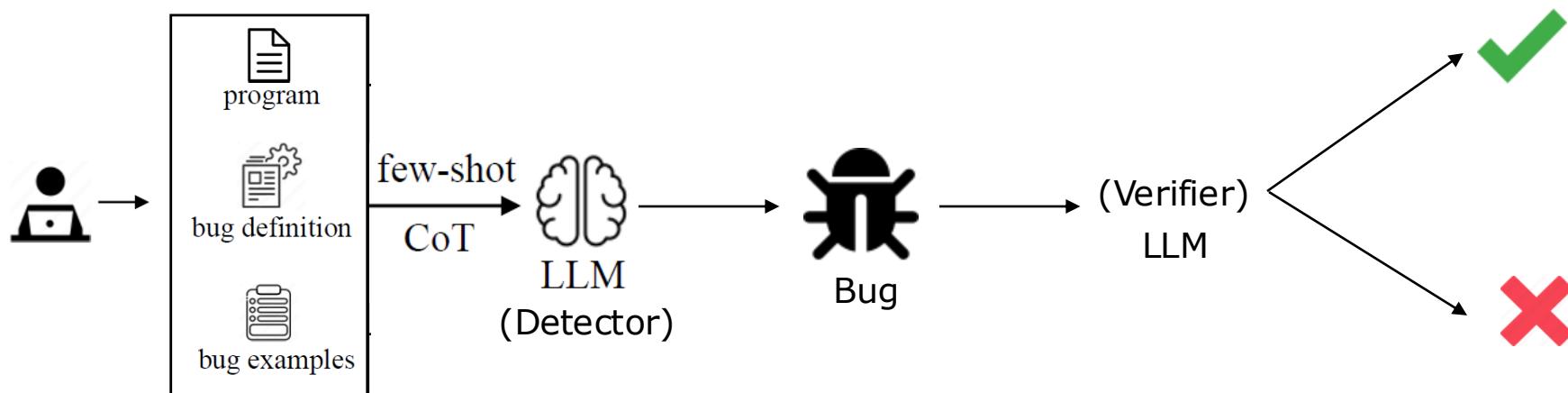
Chengpeng Wang, et al. Sanitizing Large Language Models in Bug Detection with Data-Flow. EMNLP Findings 2024

Chengpeng Wang, et al. LLMDFA: Analyzing dataflow in code with large language model. NeurIPS 2024

Jinyao Guo*, Chengpeng Wang*, et al. REPOAUDIT: An Autonomous LLM-Agent for Repository-Level Code Auditing. ICML 2025

Few-shot CoT Prompting-based Bug Detection

- Examine the detection results using LLMs as verifier to improve the precision
- The LLM-based verifier many still hallucinates
 - Key reason: LLM responses in natural languages are ambiguous.
 - Consequence: Causing FNs and leaving FPs



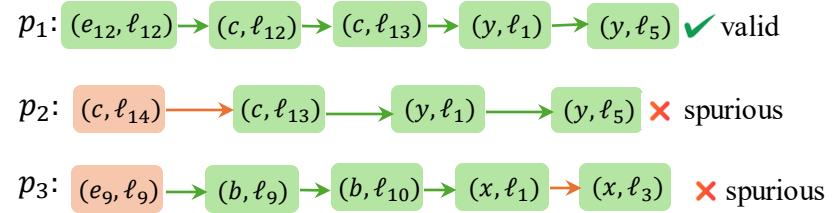
Key Idea: Make LLM Response Verifiable

- Generate bug proofs: **Data-flow paths** from sources to sinks
- Examine data-flow paths via **divide-and-conquer**
 - **Data** property: Whether start/end values conform to the forms of sources/sinks
 - **Flow** property: Whether the faulty value can be propagated along the path in each step

```
1 public static void bar(int x, int y){  
2     if (x != 0)  
3         return (y * 1.0 / x);  
4     else  
5         return (x * 1.0 / y); //bug  
6 }  
7 public static void main(){  
8     int a = 0; //zero  
9     int b = parseInt("123");  
10    System.out.println(bar(b, a));  
11    String arg = args[0];  
12    int c = parseInt(arg); //potential zero  
13    System.out.println(bar(a, c));  
14    c = b;  
15    System.out.println(bar(a, c));  
16 }
```

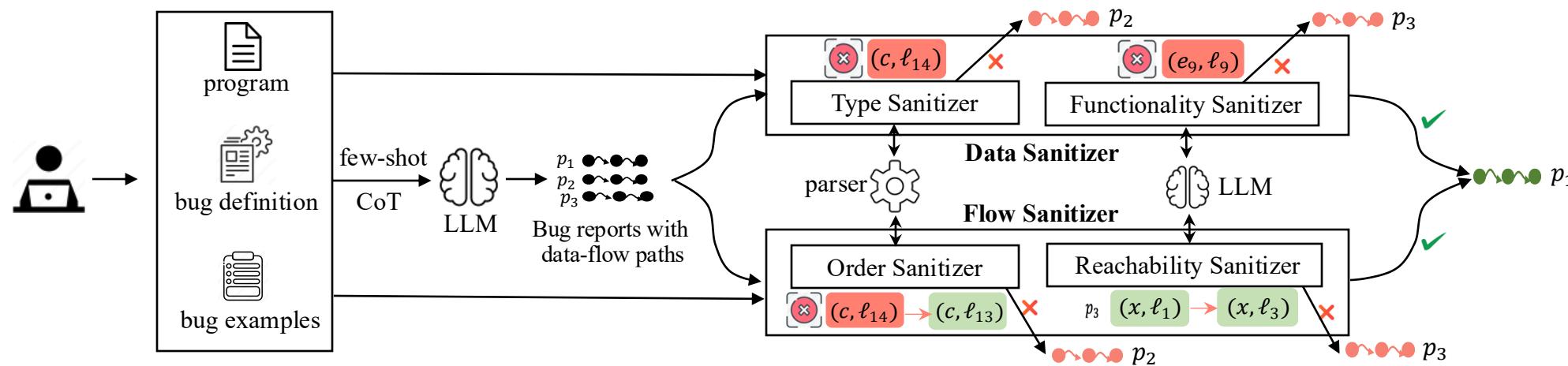
Program: [Example Program with a DBZ bug]
Explanation: v is assigned with 0 at line 3. Then u is initialized with v at line 7 and used as a divisor at line 9, causing a DBZ bug.
Dataflow path: $[(v, \ell_3), (u, \ell_7), (u, \ell_9)]$

Few-shot Example



LLMSAN: LLM-driven Bug Detection with Sanitization

- Sanitize **data-flow paths** emitted by few-shot CoT prompting
- Four sanitizers powered by parsers and LLMs
 - Decompose the validation of **syntactic** and **semantic** properties
 - **Syntactic properties** can be perfectly validated by **parsing-based sanitizers**



Main Results: Juliet Test Suite

- Achieve comparable and even better performance than symbolic analysis
 - Symbolic analysis: Pinpoint [PLDI'18] and CodeFuseQuery [ICSE'25]
- Improve the precision of few-shot CoT prompting (FSCoT) by **21.99%** averagely

Bug Type	CWE ID
Absolute Path Traversal (APT)	CWE-23
Cross-Site Scripting (XSS)	CWE-79
OS Command Injection (OSCI)	CWE-78
Divide-by-Zero (DBZ)	CWE-369
Null Pointer Dereference (NPD)	CWE-476

Bug type

Bug Type	P (%)	R (%)
APT	98.18	54.00
XSS	89.77	79.00
OSCI	98.94	93.00
DBZ	92.68	76.00
NPD	75.56	68.00
Average	91.03	74.00

LLMSAN powered by GPT-4

Bug Type	Pinpoint		CodeFuseQuery		FSCoT	
	P (%)	R (%)	P (%)	R (%)	P (%)	R (%)
APT	100.00	81.00	84.21	64.00	54.00	54.00
XSS	100.00	47.00	91.86	79.00	79.00	79.00
OSCI	100.00	31.00	87.10	54.00	93.00	93.00
DBZ	92.65	63.00	29.45	81.00	68.47	76.00
NPD	87.36	76.00	35.78	73.00	50.72	70.00
Average	96.00	59.60	65.68	70.20	69.04	74.40

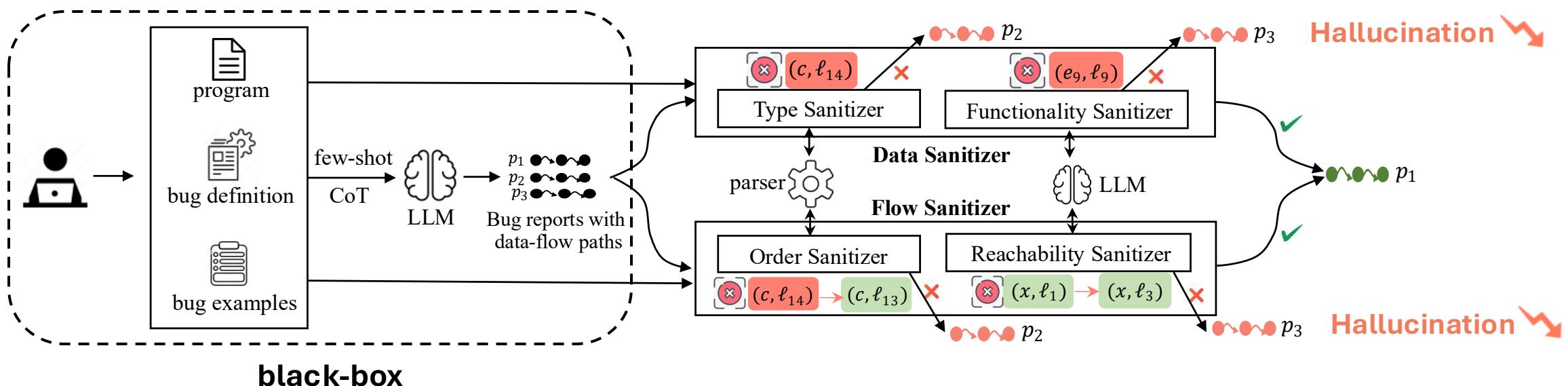
Baselines

Qingkai S., et al. Pinpoint: fast and precise sparse value flow analysis for million lines of code. PLDI 2018

Qingkai S., et al. CodeFuseQuery: Datalog-Based Language-Agnostic Change Impact Analysis for Microservices. ICSE 2025

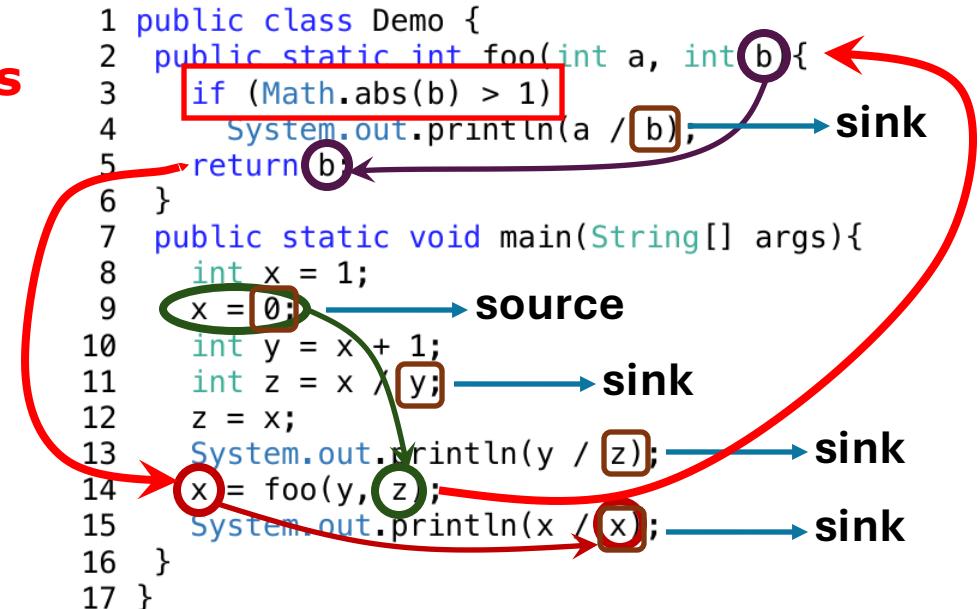
LLMSAN: Discussion

- End-to-end prompting in the detection phase
 - Pro: Compilation-free and easy to customize. **High precision.**
 - Con: **Low recall**, e.g., GPT-3.5 powered LLMSAN misses all the DBZ bugs
- How to improve recall: Avoid black-box detection



Key Idea : Break Down Call Graph

- LLM Hallucination caused by **non-linear structures**
 - Lengthy source code organized by **call graph**
- LLMDFA: Summary-based inter-procedural analysis
 - A function is more linear than the whole program
- Program decomposition
 - 👉 • Source/sink extraction
 - 👉 • Dataflow summarization
 - Path feasibility validation



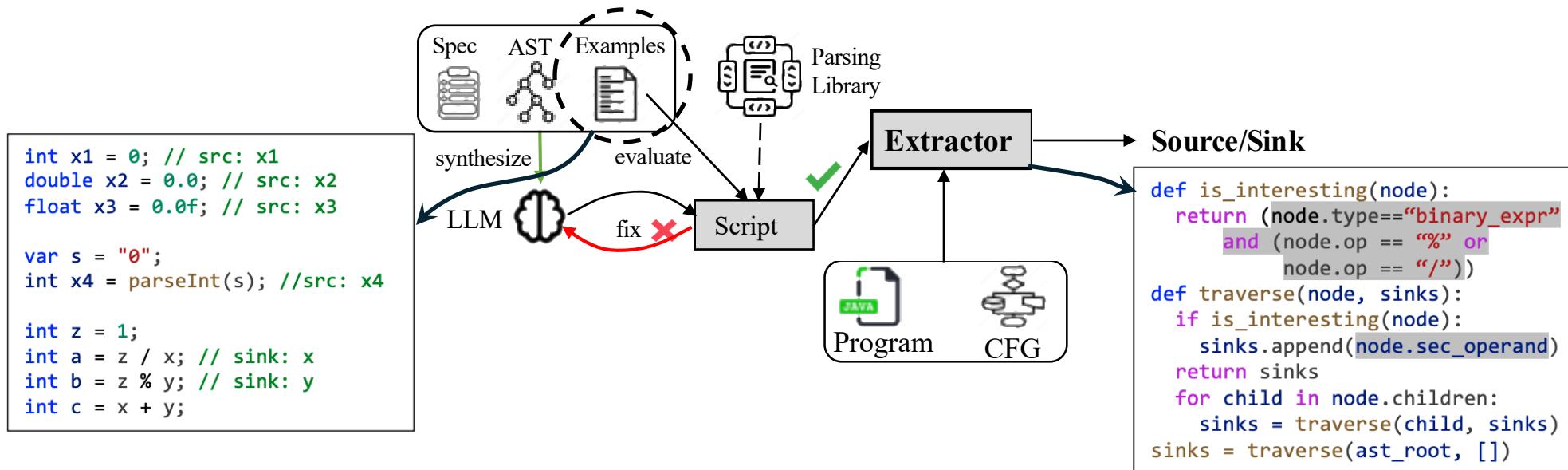
1 source -> argument

2 parameter -> return value

3 output value -> sink

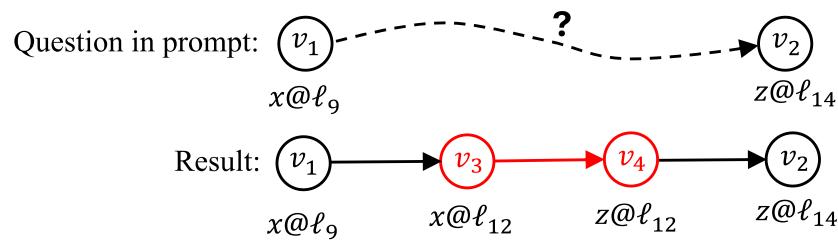
Source/Sink Extraction in LLMDFA

- Extract sources/sinks with LLMs? No! Hallucinatory and Expensive!
- Our design: Reduce the sub-problem to program synthesis problem
 - Synthesize the parsing-based extractors with LLMs
 - Apply the extractors as tools to code base
 - **No hallucination and one-time effort**



Dataflow Summarization in LLMDFA

- Our design: Summarize critical data-flow facts (v_1, v_2) in single functions via few-shot CoT
 - v_1 : sources, parameters, and output values
 - v_2 : sinks, arguments, and return values
 - Sources and sinks identified by synthesized extractors

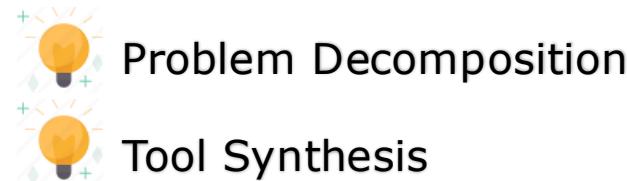
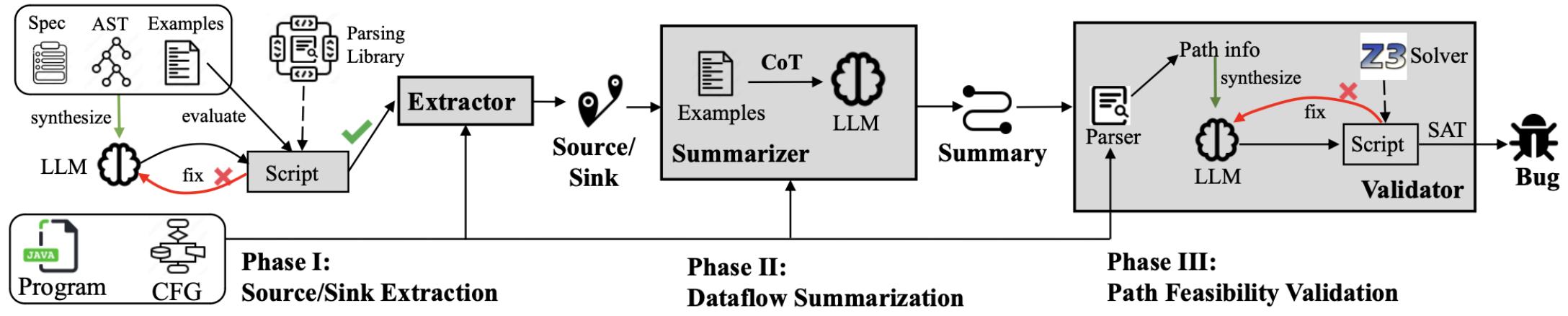


```
1 public class Demo {  
2     public static int foo(int a, int b){  
3         if (Math.abs(b) > 1)  
4             System.out.println(a / b);  
5         return b;  
6     }  
7     public static void main(String[] args){  
8         int x = 1;  
9         x = 0;  
10        int y = x + 1;  
11        int z = x / y;  
12        z = x;  
13        System.out.println(y / z);  
14        x = foo(y, z);  
15        System.out.println(x / x);  
16    }  
17 }
```

Annotations on the code:

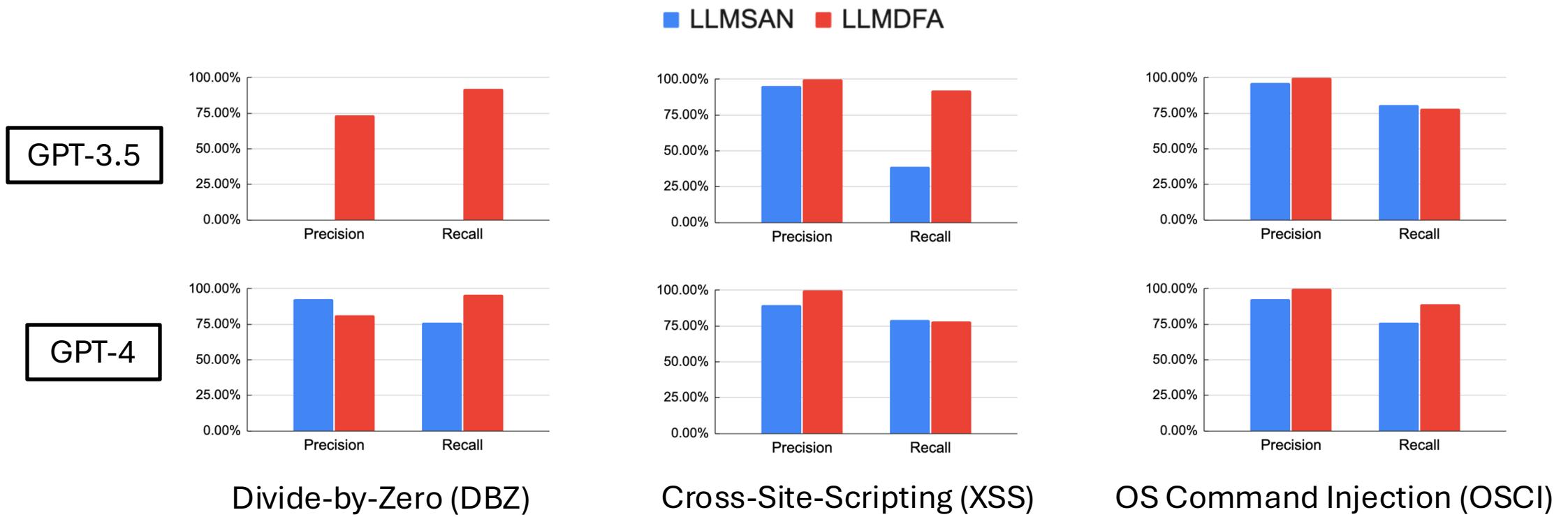
- A purple circle highlights variable `b` in the assignment `return b;`. A purple arrow points from this circle to the `b` in the parameter list of the `foo` call at line 14, with the label **parameter -> return value**.
- A green circle highlights variable `x` in the assignment `x = 0;`. A green arrow points from this circle to the `x` in the argument list of the `foo` call at line 14, with the label **source -> argument**.
- A red circle highlights variable `x` in the final `System.out.println` statement at line 15. A red arrow points from this circle to the `x` in the sink expression `x / x`, with the label **output value -> sink**.

LLMDFA: Analyzing Data-flow with LLMs



Evaluation Result: Comparison with LLMSAN

- LLMSAN vs LLMDFA powered by GPT-3.5 and GPT-4



LLMDFA: Discussion

-  High precision and recall
-  High time/token/financial cost
 - A large number of sources/sinks
 - A large number of functions
 - A large number of function calls
-  Restrictive bug types
 - Only target source-must-not-reach-sink bugs
 - Unable to detect memory leak-style bugs

```
1 public class Demo {  
2     public static int foo(int a, int b){  
3         if (Math.abs(b) > 1)  
4             System.out.println(a / b);  
5         return b  
6     }  
7     public static void main(String[] args){  
8         int x = 1;  
9         x = 0;   
10        int y = x + 1;   
11        int z = x / y;   
12        z = x;  
13        System.out.println(y / z);  
14        x = foo(y, z);   
15        System.out.println(x / x);   
16    }  
17 }
```

Annotations:

- 2 parameter -> return value
- 1 source -> argument
- 3 output value -> sink

RepoAudit: LLM-agent Auditing Code Repo as Human

- Repository-level All-Path Reasoning
 - Memory
 - Store **data-flow facts** in **different intra-procedural paths**
 - Achieve the path sensitivity
 - Tool using
 - (Symbolic) Retrieval: Source/Sink, Caller/Callee
 - (Neural) Function interpreter: Intra-procedural data-flow facts in each intra-procedural path
 - (Neuro-Symbolic) Path validator: Validate control flow order and path conditions
 - Planning
 - Start from *functions containing sources*
 - Search for sinks by exploring callers and callees **on demand** for better **scalability**



REPOAUDIT: An Autonomous LLM-Agent for Repository-Level Code Auditing . <https://repoaudit-home.github.io/>

Memory Corruption Detection in Real-world Systems

- Bug Types
 - Null Pointer Dereference (NPD)
 - Memory Leak
 - Use After Free
- Setting
 - Claude 3.5 Sonnet
 - Reproduce reported old bugs
 - Detect potential new bugs
- Result
 - Precision: **65.52%**
 - Recall: Reproduce all old bugs.
 - **26 new bugs**
 - Cost: **0.58 hours and \$3.13**

NPD					
Repo	TP	FP	Old	New	LoC
sofa-pbrpc	10	2	1	9	40,723
ImageMagick/MagickCore	7	0	7	0	242,555
coturn/src/server	2	2	1	1	8,976
libfreenect	7	5	1	6	37,582
openldap	2	4	1	1	442,955
Total	28	13	11	17	
Memory Leak					
Repo	TP	FP	Old	New	LoC
libsass	4	3	1	3	40,934
memcached	4	5	1	3	14,654
linux/driver/net	1	0	1	0	914,025
linux/sound	1	1	1	0	1,378,262
linux/mm	1	2	1	0	171,721
Total	11	11	5	6	
Use After Free					
Repo	TP	FP	Old	New	LoC
Redis	1	0	1	0	179,723
linux/drivers/peci	1	0	1	0	2,130
shadowsocks-libev	1	0	1	0	71,080
wabt-tool	0	0	0	0	3,214
icu/icu4c/source/i18n	5	3	2	3	220,359
Total	8	3	5	3	

Fixed: 80

Confirmed: 90

Reproduced: 27

Pending: 17

ID: 1 uber / h3 Search: Type / to search +

Code Issues Pull requests Discussions Actions Projects Security Insights

Fix a memory leak bug in function polygonStringToGeoPolygon #976

Merged isaacbrodsky merged 1 commit into `uber:master` from `mugitya03:master` on Feb 27

Conversation 2 Commits 1 Checks 36 Files changed 1 +1 -0

mugitya03 commented on Feb 26

The pointer `verts` at line 14 of function `polygonStringToGeoPolygon` is not freed before the function returns at line 34. Thus there is a memory leak bug.

```
LatLn *verts = calloc(numVerts, sizeof(LatLn));  
  
if (curDepth > 4) {  
    // This is beyond the depth for a valid input, so we abort early  
    return E_FAILED;  
}
```

Reviewers: dfellis, isaacbrodsky

Assignees: No one assigned

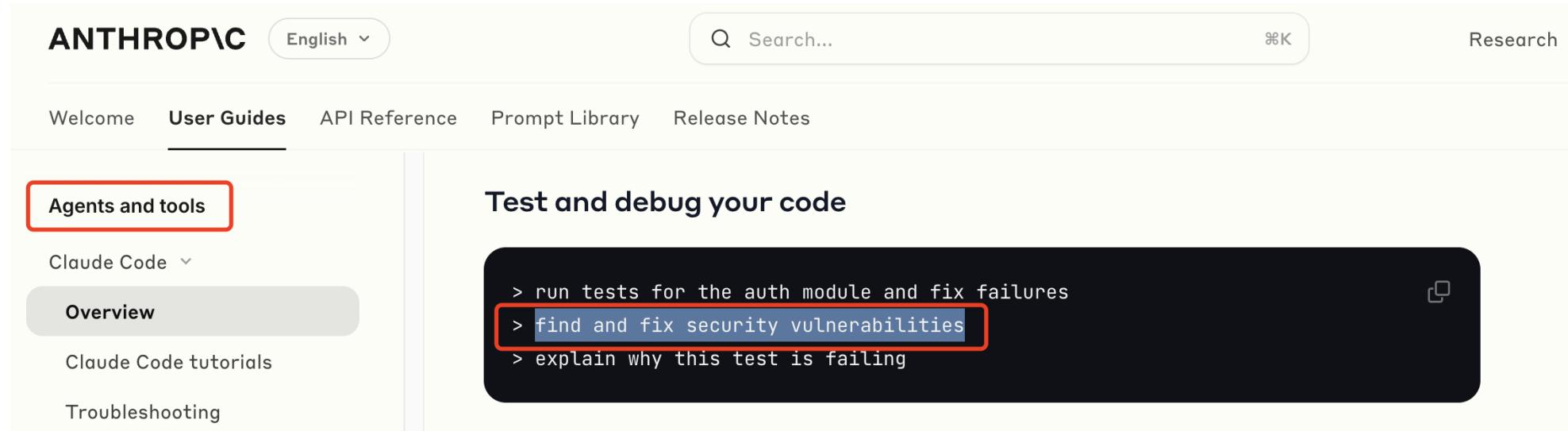
Labels: None yet

Projects: None yet

Fix a memory leak bug in function polygonStringToGeoPolygon ... Verified ✓ 12f9d27

Industry Effort: Coding Agent powered by Claude-3.7

- Find and fix security vulnerabilities
 - Local bugs: Grep-like search + Guess semantics from symbol names
 - Unable to support the inter-procedural data-flow analysis
 - Miss many true bugs detected by RepoAudit but report many false positives

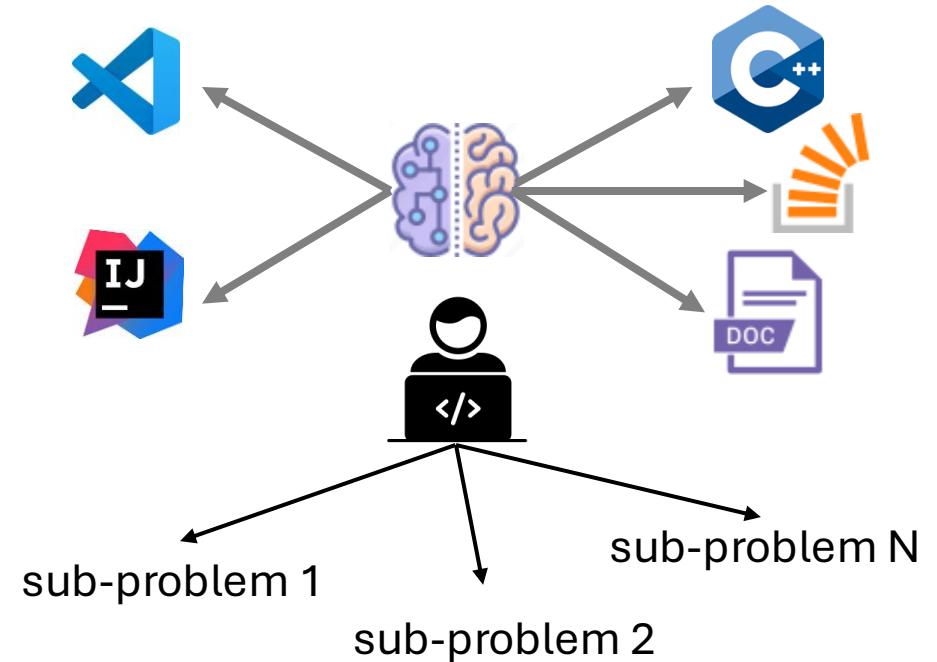


Outline

- Static Bug Detection
- LLM-driven Data-flow Bug Detection
 - LLMSAN
 - LLMDFA
 - RepoAudit
- Neuro-Symbolic Static Analysis

Neuro-Symbolic Analysis: Analyzing Code as Human

- Problem decomposition
 - Multiple manageable sub-problems
- Using tools in IDE
 - String search / structural search
 - Usage of functions, variables, and classes
- Experience-driven manual analysis
 - Language standards
 - Discussions in Stack Overflow
 - Library documentation



How to Build Neuro-Symbolic Static Analysis

- Explore diverse neuro-symbolic architectures
 - Revisit traditional static analysis algorithms from neuro-symbolic perspective
 - Example: Summary-based static analysis (LLMDFA), demand-driven path exploration (RepoAudit)
- Collect and design external domain experts
 - Integrated as symbolic components
 - Example: SMT solvers (LLMDFA) and self-defined tools (LLMSAN & RepoAudit)
- Unleash the power of LLMs
 - Evaluate the LLMs upon different tasks in empirical study and build domain-specific models
 - Example: Program synthesis for tool synthesis (LLMDFA), and function interpreter (RepoAudit)

More Instances

- Support more **diverse bug types**
 - Bugs that can not be well-defined/formulated, such as **functional bugs**
 - Insight: LLMs can understand the bug spec and reason as human
- Support **multi-modal bug detection**
 - Non-code artifacts, such as **docs and bug reports**, offer informal semantic properties [FSE 2024]
 - Insight: LLMs can derive semantic properties from docs and apply them in downstream analysis
- Support **customizable static analysis workflow**
 - **Define an AI agent using DSL**, such as Datalog, for different clients, e.g., slicing
 - Insight: Datalog uniformly organizes the properties generated by LLMs and derive the desired one

Chengpeng Wang, et al. DAInfer: Inferring API Aliasing Specifications from Library Documentation via Neurosymbolic Optimization. FSE 2024

Towards Neuro-Symbolic Static Analysis

Symbolic Static Analysis

- Rule-based expert system
 - 👍 ➤ Explainable
 - 👍 ➤ Deterministic
 - 👎 ➤ Limited applicability
 - 👎 ➤ Restricted usability
 - 👎 ➤ Single modality (code only)



Neural Static Analysis

- Data-driven black box
 - 👍 ➤ Extensive applicability
 - 👍 ➤ Flexible usability
 - 👍 ➤ Multi-modality (code, doc, etc)
 - 👎 ➤ Unexplainable
 - 👎 ➤ Nondeterministic
 - 👎 ➤ Hallucinatory



Neuro-Symbolic Static Analysis

- LLM agent
 - 👍 ➤ (Relatively) Explainable
 - 👍 ➤ (Relatively) Deterministic
 - 👍 ➤ Extensive applicability
 - 👍 ➤ Flexible usability
 - 👍 ➤ Multi-modality

LLMSAN **LLMDFA**
RepoAudit

Conclusion

- Static analysis is critical for **software reliability** in the LLM era.
- LLMs are **not silver bullets** due to hallucinations when dealing with non-linear contexts.
- Developing **neuro-symbolic static analysis** techniques holds great potential.

Q&A

- Chengpeng Wang
- Homepage: <https://chengpeng-wang.github.io/>
- Email: wang6590@purdue.edu
- RepoAudit: <https://repoaudit-home.github.io/>

Comparison with Cursor

- Agent: Cursor with Claude-3.7-Sonnet Max
- Bug Type: Null Pointer Dereference (NPD)
- Prompting settings
 - Forward Analysis: Target NULL values in a single file
 - Backward Analysis: Target dereferences in a single file
 - Whole Repo Analysis: Target all contexts and dereferences in the repo

Hashtable.c — RepoAudit-Plus [SSH: homepd4]

Hashtable.c X New chat

REPOAUDIT-PLUS [SSH: ...]

benchmark > Cpp > htop > C Hashtable.c

Christian Götsche, 13 months ago | 7 authors (Christian Götsche and others)

```

1  /*
2  htop - Hashtable.c      Hisham Muhammad, 13 years ago • major header cleanup ...
3  (C) 2004-2011 Hisham H. Muhammad
4  Released under the GNU GPLv2+, see the COPYING file
5  in the source distribution for its full text.
6  */
7
8  #include "config.h" // IWYU pragma: keep
9
10 #include "Hashtable.h"
11
12 #include <assert.h>
13 #include <stdint.h>
14 #include <stdlib.h>
15 #include <string.h>
16
17 #include "CRT.h"
18 #include "Macros.h"
19 #include "XUtils.h"
20
21 #ifndef NDEBUG
22 #include <stdio.h>
23 #endif
24
25
26 typedef struct HashtableItem_ {
27     ht_key_t key;
28     size_t probe;
29     void* value;
30 } HashtableItem;
31
32 struct Hashtable_ {
33     size_t size;
34     HashtableItem* buckets;
35     size_t items;
36     bool owner;
37 };
38
39
40 #ifndef NDEBUG
41
42 static void Hashtable_dump(const Hashtable* this) {
43     fprintf(stderr, "Hashtable %p: size=%zu items=%zu owner=%s\n",
44 
```

Please detect all the NPD bugs in the project benchmark/Cpp/htop that are caused by the NULL values produced in the file Hashtable.c. The dereference points of the NULL values can be located in the files different from the file Hashtable.c. You should also detect them.

Agent MI claude-3.7-sonnet

Send

Maximum intelligence, context, and thinking
\$0.05 per request and per tool use

claude-3.7-sonnet MAX ✓
claude-3.7-sonnet
gemini-2.5-pro-exp-03-25
gemini-2.5-pro MAX
o4-mini
o3

Prompting Setting I: Forward Analysis

Whether NULL values in a specific file can cause NPD bugs within the code repo.

SSH: homepd4 e8236c43* Launchpad 0 0 1

Hisham Muhammad, 13 years ago Ln 2, Col 19 Spaces: 3 UTF-8 LF {} C Cursor Tab



REPOAUDIT-PLUS [SSH: HOMEPD4]

- ✓ benchmark
- ✓ Cpp
- ✓ htop
- ✓ linux
 - C IOPriorityPanel.c
 - C IOPriorityPanel.h
 - C LibNI.c
 - C LibNI.h
 - C LibSensors.c
 - C LibSensors.h
 - C LinuxMachine.c
 - C LinuxMachine.h
 - C LinuxProcess.c
 - C LinuxProcess.h
 - C LinuxProcessTable.c
 - C LinuxProcessTable.h
 - C Platform.c
 - C Platform.h
 - C PressureStallMeter.c
 - C PressureStallMeter.h
 - C ProcessField.h
 - C SELinuxMeter.c
 - C SELinuxMeter.h
 - C SystemdMeter.c
 - C SystemdMeter.h
 - C ZramMeter.c
 - C ZramMeter.h
 - C ZramStats.h
 - C ZswapStats.h
- > netbsd
- > openbsd
- ✓ pcp
 - > columns
 - > meters

> OUTLINE

> TIMELINE

SSH: homepd4

e8236c43



Launchpad



0



0



1

Please detect all the NPD bugs in the project benchmark/Cpp/htop that are caused by the NULL values produced in the file Hashtable.c. The dereference points of the NULL values can be located in the files different from the file Hashtable.c. You should also detect them.

⌚ Thought for 5 seconds

I'll search for potential NULL pointer dereference (NPD) bugs related to Hashtable.c in the htop project.

🔍 Searched codebase for "Hashtable_get dereference null" (benchmark/Cpp/htop/)

Let's look for more potential usage of Hashtable functions, especially focusing on cases where NULL returns might be dereferenced.

🔍 Searched codebase for "Hashtable_remove null dereference" (benchmark/Cpp/htop/)

Let's search for more instances where we might find usages of the NULL values returned by Hashtable functions:

🔍 Searched codebase for "ProcessTable_getProcess dereference" (benchmark/Cpp/htop/)

Let's search for other uses of `DynamicColumn_lookup` and `UsersTable_getRef` since they can also return NULL values:

🔍 Searched codebase for "DynamicColumn_lookup dereference" (benchmark/Cpp/htop/)

Now let's search for usages of `UsersTable_getRef` that might dereference its return NULL value:

🔍 Searched codebase for "UsersTable_getRef dereference user" (benchmark/Cpp/htop/)

search by function

Let's examine the `Process.h` file and specifically check where the `user` field is used to understand potential dereference points:

⌚ Read `Process.h`

Let's search for places where `proc->user` might be dereferenced or used:

🔍 Grepped codebase for "proc->user"

search by value via grep

Let's examine if the `Process.c` file has logic that might dereference the `user` field:

⌚ Read `Process.c` L1-500

Let's search for usage of `Process_writeField` or `Process_rowWriteField` to understand where users might be dereferenced:

⌚ Read `Process.c` L530-1029

Based on my analysis of the code, I've found several potential null pointer dereference (NPD) bugs related to Hashtable.c null values. Here's a list of the bugs found:

1. Bug in `Process.c`'s `Process_writeField` function:

⌚ At line ~640, `this->user` is dereferenced directly without a null check when displaying the `USER` field.

@ Add context

Plan, search, build anything

⌚ Agent



claude-3.7-sonnet

Send

Reasoning steps after planning

Cursor Tab





Chat X



REPOAUDIT-PLUS [SSH: HOMEPD4]

- ✓ benchmark
- ✓ Cpp
- ✓ htop
- ✓ linux
 - C IOPriorityPanel.c
 - C IOPriorityPanel.h
 - C LibNL.c
 - C LibNL.h
 - C LibSensors.c
 - C LibSensors.h
 - C LinuxMachine.c
 - C LinuxMachine.h
 - C LinuxProcess.c
 - C LinuxProcess.h
 - C LinuxProcessTable.c
 - C LinuxProcessTable.h
 - C Platform.c
 - C Platform.h
 - C PressureStallMeter.c
 - C PressureStallMeter.h
 - C ProcessField.h
 - C SELinuxMeter.c
 - C SELinuxMeter.h
 - C SystemdMeter.c
 - C SystemdMeter.h
 - C ZramMeter.c
 - C ZramMeter.h
 - C ZramStats.h
 - C ZswapStats.h
- > netbsd
- > openbsd
- ✓ pcp
 - > columns
 - > meters

> OUTLINE

> TIMELINE

SSH: homepd4 e8236c43 Launchpad 0 △ 0 A 1

Cursor Tab

Based on my analysis of the code, I've found several potential null pointer dereference (NPD) bugs related to Hashtable.c null values. Here's a list of the bugs found:

1. Bug in Process.c's `Process_writeField` function:

- At line ~640, `this->user` is dereferenced directly without a null check when displaying the USER field.
- `this->user` is populated from `UsersTable_getRef()` which can return NULL if both the Hashtable lookup fails and the `getpwuid()` call returns NULL.
- This would lead to a null pointer dereference when trying to display user names for processes with unknown UIDs.

2. Bug in DynamicScreen.c's `DynamicScreen_lookup` function:

- At line ~61, the function returns `screen->name` after calling `Hashtable_get(screens, key)` without checking if `screen` is NULL.
- If the key doesn't exist in the hashtable, `Hashtable_get` returns NULL, and then trying to access `screen->name` would cause a null pointer dereference.

3. Bug in Instance.c's `Instance_writeField` function:

- At line ~69, the code gets a `DynamicColumn` from a hashtable with `Hashtable_get(settings->dynamicColumns, field)` and directly casts it to `PCPDynamicColumn` without a null check.
- If the column isn't found in the hashtable, dereferencing `cp` would cause a null pointer dereference.

4. Bug in ProcessTable.c's `ProcessTable_getProcess` function:

- When a process isn't found in the hashtable, a new one is created, but there's no null check after the constructor call before dereferencing `proc->cmdline` at line ~31-32.
- If the constructor returns NULL (unlikely but potentially possible), this would cause a null pointer dereference.

5. Bug in Table.c's `Table_removeIndex` function:

- At line ~91, there's a call to `Hashtable_remove(this->table, rowid)` and potentially using the result without checking if it's NULL.
- In this case, although the return value from `Hashtable_remove` is not used, there's still a risk if the code is modified in the future to use this value.

These bugs all stem from functions that use values from Hashtable.c without proper null checks. In particular, the most critical ones are:

1. The user field dereference in `Process_writeField`
2. The `screen->name` dereference in `DynamicScreen_lookup`
3. The dynamic column dereference in `Instance_writeField`

These functions directly use values returned from `Hashtable_get` without checking if they're NULL, which can lead to crashes when invalid keys are used.

False Positives

@ Add context

Plan, search, build anything

Agent claudie-3.7-sonnet

Send

Chat X

@ Add context
Detect the NPD bugs in the file `benchmark/Cpp/htop/pcp/PCPDynamicScreen.c`

Agent 🤖 clade-3.7-sonnet Send

Maximum intelligence, context, and thinking
\$0.05 per request and per tool use

Auto-select Thinking

clade-3.7-sonnet MAX ✓
clade-3.7-sonnet
gemini-2.5-pro-exp-03-25
gemini-2.5-pro MAX
o4-mini
o3

Prompting Setting II: Backward Analysis

Whether the pointer dereferences in a specific file can cause NPD bugs

> OUTLINE Past chats >

> TIMELINE

SSH: homepd4 e8236c43 Launchpad 0 △ 0 A 1 Cursor Tab



REPOAUDIT-PLUS [SSH: HOMEPD4]

benchmark

Cpp

htop

> netosa

> openbsd

pcp

> columns

> meters

> screens

C InDomTable.c

C InDomTable.h

C Instance.c

C Instance.h

C Metric.c

C Metric.h

C PCPDynamicColumn.c

C PCPDynamicColumn.h

C PCPDynamicMeter.c

C PCPDynamicMeter.h

C PCPDynamicScreen.c

C PCPDynamicScreen.h

C PCPMachine.c

C PCPMachine.h

C PCPProcess.c

C PCPProcess.h

C PCPProcessTable.c

C PCPProcessTable.h

C Platform.c

C Platform.h

C ProcessField.h

> scripts

> solaris

> unsupported

> zfs

> editorconfig

> OUTLINE

> TIMELINE

Detect the NPD bugs in the file `benchmark/Cpp/htop/pcp/PCPDYNAMICSCREEN.C`

Thought for 2 seconds

Let me examine the file for Null Pointer Dereference bugs.

Read PCPDYNAMICSCREEN.C

Let me examine this file for potential Null Pointer Dereference (NPD) bugs. I'll analyze the code methodically to identify instances where pointers might be dereferenced without proper null checks.

Here are the NPD bugs I've identified:

1. Line 54-55:

```
c c
PCPDYNAMICSCREEN* screen = Hashtable_get(screens->table, i);
if (!screen)
    return;
```

This is in `PCPDYNAMICSCREENS_appendDynamicColumns()`. The function immediately returns if `screen` is NULL, but doesn't check whether any operations on `screens` are safe beforehand. If `screens` or `screens->table` is NULL, this will cause an NPD.

2. Line 89-94:

```
c c
column = screen->columns[i];
if (String_eq(column->metricName, metricName)) {
    free(metricName);
    return column;
}
```

In `PCPDYNAMICSCREEN_lookupMetric()`, if `column` is NULL, dereferencing `column->metricName` would cause an NPD.

3. Line 256:

```
c c
if (xdgConfigHome)
    path = String_cat(xdgConfigHome, "/htop/screens/");
```

There's no check to verify that `String_cat()` returns a non-NULL value before using `path` in the following function calls.

4. Line 354:

@ Add context

Plan, search, build anything

Agent [] ^ claudie-3.7-sonnet ^

False Positives

Lack of calling contexts



REPOAUDIT-PLUS [SSH: HOMEPD4]

- ✓ benchmark
- ✓ Cpp
- ✓ htop
- ✓ netosa
- > openbsd
- ✓ pcp
- > columns
- > meters
- > screens
- C InDomTable.c
- C InDomTable.h
- C Instance.c
- C Instance.h
- C Metric.c
- C Metric.h
- C PCPDynamicColumn.c
- C PCPDynamicColumn.h
- C PCPDynamicMeter.c
- C PCPDynamicMeter.h
- C PCPDynamicScreen.c
- C PCPDynamicScreen.h
- C PCPMachine.c
- C PCPMachine.h
- C PCPProcess.c
- C PCPProcess.h
- C PCPProcessTable.c
- C PCPProcessTable.h
- C Platform.c
- C Platform.h
- C ProcessField.h
- > scripts
- > solaris
- > unsupported
- > zfs
- editorconfig

> OUTLINE

> TIMELINE

Apply

In `PCPDYNAMICSCREEN_LOOKUPMETRIC()`, if `column` is NULL, dereferencing `column->metricName` would cause an NPD.

3. Line 256:

Apply

```
if (xdgConfigHome)
    path = String_cat(xdgConfigHome, "/htop/screens/");
```

There's no check to verify that `String_cat()` returns a non-NULL value before using `path` in the following function calls.

4. Line 354:

Apply

```
PCPDYNAMICSCREEN* dynamicScreen = Hashtable_get(screens, key);
if (!screen)
    return;
```

True Positive

There's a logic error here - the check should be for `dynamicScreen`, not `screen`. If `dynamicScreen` is NULL, the function continues and accesses `dynamicScreen->totalColumns` in the for loop, which would cause an NPD.

5. Line 347-349:

Apply

```
bool success;
unsigned int key;
success = DynamicScreen_search(screens, screen, &key);
```

There is no check to ensure that `screens` or `screen` is not NULL before passing them to `DynamicScreen_search()`.

These are the most evident null pointer dereference bugs in the provided code. The code would benefit from additional null checks, especially when dealing with functions that may return NULL values or when accessing pointer members.

False Positive

Lack of calling contexts

@ Add context

Plan, search, build anything

Agent claude-3.7-sonnet

Send

htop.c — RepoAudit-Plus [SSH: homepd4]

REPOAUDIT-PLUS [SSH: HOMEPD4]

benchmark > Cpp > htop > C htop.c

Nathan Scott, 2 years ago | 5 authors (Hisham Muhammad and others)

```
1  /* Hisham Muhammad, 19 years ago * Initial import.
2  htop - htop.c
3  (C) 2004-2011 Hisham H. Muhammad
4  (C) 2020-2021 htop dev team
5  Released under the GNU GPLv2+, see the COPYING file
6  in the source distribution for its full text.
7  */
8
9  #include "config.h" // IWYU pragma: keep
10
11 #include "CommandLine.h"
12
13
14 const char* program = PACKAGE;
15
16 int main(int argc, char** argv) {
17     return CommandLine_run(argc, argv);
18 }
19
```

@ Add context
Please detect NPD bugs in the project benchmark/Cpp/htop

Agent AI (claude-3.7-sonnet)

Maximum intelligence, context, and thinking
\$0.05 per request and per tool use

Auto-select Thinking claude-3.7-sonnet MAX ✓
claude-3.7-sonnet gemini-2.5-pro-exp-03-25 gemini-2.5-pro MAX o4-mini o3

Prompting Setting III: Whole Repo Analysis

Detect all the potential NPD bugs in the repo

OUTLINE

TIMELINE

SSH: homepd4 e8236c43 Launchpad 0 △ 0 ⌂ 0

Hisham Muhammad, 19 years ago Ln 1, Col 3 Spaces: 3 UTF-8 LF {} C Cursor Tab



REPOAUDIT-PLUS [SSH: HOMEPD4]

- benchmark
- Cpp
- htop
- netosa
- openbsd
- pcp
- columns
- meters
- screens
- InDomTable.c
- InDomTable.h
- Instance.c
- Instance.h
- Metric.c
- Metric.h
- PCPDynamicColumn.c
- PCPDynamicColumn.h
- PCPDynamicMeter.c
- PCPDynamicMeter.h
- PCPDynamicScreen.c
- PCPDynamicScreen.h
- PCPMachine.c
- PCPMachine.h
- PCPProcess.c
- PCPProcess.h
- PCPProcessTable.c
- PCPProcessTable.h
- Platform.c
- Platform.h
- ProcessField.h
- scripts
- solaris
- unsupported
- zfs
- editorconfig

> OUTLINE

> TIMELINE

Please detect NPD bugs in the project benchmark/Cpp/htop

⌚ Thought for 2 seconds

I'll detect null pointer dereference bugs in the htop project. Let me explore the project structure first.

⌚ Listed 100 items in benchmark/Cpp/htop

I'll search for potential NULL pointer dereference bugs in the codebase. Let me look for common patterns that might indicate NPD issues.

⚠ Model provided invalid arguments to grep tool.

Let me modify my search query to find potential NPD bugs:

⌚ Grepped codebase for "if\s*\|\(\|s*\|\w+\|s*==\|s*NULL\|s*\|\)\[^|\{|\}\|\{|\}*\|\w+\|s*->"

Let me try a different approach and look for places where pointers are directly dereferenced without null checks:

⌚ Grepped codebase for "return\|s+\|\w+\|s*->\|\w+\|s*->[^=]*=[^=]"

Let me look for functions that might return NULL and see how they're used:

⌚ Grepped codebase for "return NULL;"

I see the search is showing results from a different project (cpv-1), not htop. Let me try to search specifically in the htop directory:

⌚ Grepped codebase for "return NULL;" (benchmark/Cpp/htop/*)

Now I'll look for some patterns of incorrect null pointer handling. Let me check places where pointers might be used without proper null checks:

⌚ Grepped codebase for "free\|(\|s*\|\w+\|s*\|\)" (benchmark/Cpp/htop/*)

Let me try a more flexible approach to find free() calls:

⚠ Model provided invalid arguments to grep tool.

Let me fix that:

⌚ Grepped codebase for "free\|s*\|\([^\n]\)*\|\" (benchmark/Cpp/htop/*)

Let me look at specific files that might contain NPD bugs. I'll check XUtils.c which had a NULL return:

⌚ Read XUtils.c

Let me also check Vector.c where we saw a NULL return:

@ Add context

Plan, search, build anything

Agent claude-3.7-sonnet

search by value via grep

Reasoning steps after planning



REPOAUDIT-PLUS [SSH: HOMEPD4]

- ✓ benchmark
- ✓ Cpp
- ✓ htop
- > netosu
- > openbsd
- ✓ pcp
- > columns
- > meters
- > screens
- C InDomTable.c
- C InDomTable.h
- C Instance.c
- C Instance.h
- C Metric.c
- C Metric.h
- C PCPDynamicColumn.c
- C PCPDynamicColumn.h
- C PCPDynamicMeter.c
- C PCPDynamicMeter.h
- C PCPDynamicScreen.c
- C PCPDynamicScreen.h
- C PCPMachine.c
- C PCPMachine.h
- C PCPProcess.c
- C PCPProcess.h
- C PCPProcessTable.c
- C PCPProcessTable.h
- C Platform.c
- C Platform.h
- C ProcessField.h
- > scripts
- > solaris
- > unsupported
- > zfs

> OUTLINE

> TIMELINE

Based on my code review, I've found several potential null pointer dereference issues in the htop project. Here are my findings:

1. Issue in `free_and_xStrdup` function in XUtils.c (line 274-279):

```
c c
void free_and_xStrdup(char** ptr, const char* str) {
    if (*ptr && String_eq(*ptr, str))
        return;

    free(*ptr);
    *ptr = xStrdup(str);
}
```

False Positive

Lack of calling contexts

The function checks if `*ptr` is non-NULL for the comparison, but still calls `free(*ptr)` without checking if `ptr` is NULL. If the caller passes a NULL `ptr`, it will cause a null pointer dereference. There is no check to ensure `ptr` itself is not NULL before dereferencing it.

2. Potential issue in `Vector_softRemove` in Vector.c (lines 246-265):

```
c c
Object* Vector_softRemove(Vector* this, int idx) {
    assert(idx >= 0 && idx < this->items);

    Object* removed = this->array[idx];
    assert(removed);
    if (removed) {
        this->array[idx] = NULL;

        this->dirty_count++;
        if (this->dirty_index < 0 || idx < this->dirty_index) {
            this->dirty_index = idx;
        }

        if (this->owner) {
            Object_delete(removed);
            return NULL;
        }
    }

    return removed;
}
```

@ Add context

Plan, search, build anything

Agent claude-3.7-sonnet

Send

REPOAUDIT-PLUS [SSH: ...] Chat PCPProcessTable.c

2. Potential issue in `Vector_softRemove` in Vector.c (lines 246-265):

```
c c
Object* Vector_softRemove(Vector* this, int idx) {
    assert(idx >= 0 && idx < this->items);

    Object* removed = this->array[idx];
    assert(removed);
    if (removed) {
        this->array[idx] = NULL;

        this->dirty_count++;
        if (this->dirty_index < 0 || idx < this->dirty_index) {
            this->dirty_index = idx;
        }

        if (this->owner) {
            Object_delete(removed);
            return NULL;
        }
    }
}
return removed;
```

False Positive

Hallucinations in control-flow order

The code asserts that `removed` is not NULL, but then checks it with an if condition. If asserts are disabled in a release build, there's potential to return a NULL pointer that's already been freed if `this->owner` is true.

3. Issue in `CGroup_filterName` in linux/CGroupUtils.c (around line 370):

```
c c
char* CGroup_filterName(const char* cgroup) {
    StrBuf_state s = {
        .buf = NULL,
        .size = 0,
        .pos = 0,
    };
}
```

@ Add context

Plan, search, build anything

Agent claudie-3.7-sonnet

Send

> OUTLINE

> TIMELINE

SSH: homepd4 e8236c43 Launchpad 0 △ 0 A 1 Cursor Tab



REPOAUDIT-PLUS [SSH: HOMEPD4]

- benchmark
- Cpp
- htop
- netosa
- openbsd
- pcp
- columns
- meters
- screens
- InDomTable.c
- InDomTable.h
- Instance.c
- Instance.h
- Metric.c
- Metric.h
- PCPDynamicColumn.c
- PCPDynamicColumn.h
- PCPDynamicMeter.c
- PCPDynamicMeter.h
- PCPDynamicScreen.c
- PCPDynamicScreen.h
- PCPMachine.c
- PCPMachine.h
- PCPProcess.c
- PCPProcess.h
- PCPProcessTable.c
- PCPProcessTable.h
- Platform.c
- Platform.h
- ProcessField.h
- scripts
- solaris
- unsupported
- zfs

editorconfig

OUTLINE

TIMELINE

3. Issue in `CGroup_filterName` in `linux/CGroupUtils.c` (around line 370):

```
c c
char* CGroup_filterName(const char* cgroup) {
    StrBuf_state s = {
        .buf = NULL,
        .size = 0,
        .pos = 0,
    };

    if (!CGroup_filterName_internal(cgroup, &s, StrBuf_putc_count)) {
        return NULL;
    }

    s.buf = xCalloc(s.pos + 1, sizeof(char));
    s.size = s.pos;
    s.pos = 0;

    if (!CGroup_filterName_internal(cgroup, &s, StrBuf_putc_write)) {
        free(s.buf);
        return NULL;
    }

    s.buf[s.size] = '\0';
    return s.buf;
}
```

If `cgroup` is `NULL`, it might lead to a null pointer dereference in the internal function. There's no check to ensure `cgroup` is non-`NULL` before using it.

These issues could potentially cause crashes when the program runs into unexpected conditions where `NULL` pointers are passed to these functions. I recommend implementing proper `NULL` checks before dereferencing pointers in these cases.

False Positive

Lack of calling contexts

@ Add context

Plan, search, build anything

Agent claudie-3.7-sonnet

Send