ClearAgent: Agentic Binary Analysis for Effective Vulnerability Detection

Xiang Chen
The Hong Kong University of Science and Technology
Hong Kong, China
xchenht@cse.ust.hk

Anshunkang Zhou
The Hong Kong University of Science and Technology
Hong Kong, China
azhouad@cse.ust.hk

Chengfeng Ye
The Hong Kong University of Science and Technology
Hong Kong, China
cyeaa@cse.ust.hk

Charles Zhang
The Hong Kong University of Science and Technology
Hong Kong, China
charlesz@cse.ust.hk

Abstract

Statically detecting vulnerabilities at the binary level is crucial for the security of Commercial-Off-The-Shelf (COTS) software when source code is not available. However, traditional methods suffer from the inherent limitations of binary translation and static analysis, which hinder their scalability for complex real-world binaries. Recent efforts that leverage Large Language Models (LLMs) for vulnerability detection are still limited by possible hallucination, inaccurate code property retrieval, and insufficient guidance.

In this paper, we propose a new agentic binary analysis framework ClearAgent, which features a novel binary interface that provides both LLM-friendly and analyzer-friendly tools to facilitate effective understanding of binary code semantics with rich context. ClearAgent works by automatically interacting with the interface and iteratively exploring for buggy binary code. For candidate bug reports, ClearAgent further tries to verify the existence of the vulnerability by constructing concrete inputs that can trigger the buggy locations.

CCS Concepts: • **Security and privacy → Software and application security**.

Keywords: Agent, Binary Analysis, Vulnerability Detection

ACM Reference Format:
Xiang Chen, Anshunkang Zhou, Chengfeng Ye, and Charles Zhang. 2025. ClearAgent: Agentic Binary Analysis for Effective Vulnerability Detection. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Language Models and Programming Languages (LMPL '25), October 12–18, 2025, Singapore, Singapore.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3759425.3763397

LMPL '25, Singapore, Singapore

ACM ISBN 979-8-4007-2148-9/25/10
https://doi.org/10.1145/3759425.3763397

1 Introduction

The security of binary code has been critically important as it forms the operational backbone of virtually all digital systems, such as smartphones and edge devices. Therefore, vulnerabilities in those binaries could lead to devastating consequences such as the loss of lives and money. Static binary analysis [7, 45, 67] is one of the most effective techniques for detecting vulnerabilities in binaries. It generally works by translating binary code into analyzable intermediate representations (IRs) [27, 33, 50], which are then analyzed by rigorous static analyzers [44] or further decompiled to pseudocode for human inspection [1, 3, 5, 17].

Despite years of academic research and industrial development, mainstream static binary analyzers still suffer from three limitations when analyzing real-world binaries. First, due to information loss during compilation, accurate binary translation is undecidable [13, 26], which could further affect the static analysis results. For example, soundly resolving indirect function calls at the binary level is still challenging [14, 25], which results in buggy functions missed in the callgraph and further ignored in the analysis phase without manual exploration. Second, real-world binaries might involve complex runtime semantics that require considerable manual effort to reason about. For example, data flow via inter-process communication (IPC) channels, such as environment variables, is common in IoT firmware [9, 39]. But most existing vanilla binary analyzers cannot identify such data flow without additional case-by-case heuristics. Third, they may still suffer from inherent limitations of the underlying static analysis, such as the well-known problem of path explosion [39], limited context depth [29], etc.

Recently, Large Language Models (LLMs) have demonstrated strong capabilities in binary analysis. Researchers have focused on using prompt engineering with either specially trained models or general-purpose models in reverse engineering (RE) tasks such as lifting [10], type recovery [22, 40, 55, 56], function summary [8, 63], taint analysis [30], and decompilation [19, 49, 53]. However, unlike the above RE tasks that focus on specific code fragments, binary-level vulnerability detection requires a comprehensive understanding of the entire binary code. Therefore, directly utilizing LLMs could be inaccurate because they usually rely on chain-of-thought reasoning, where the model uses its internal representations to generate thoughts and is not grounded in the external world. As a result, the reasoning process could lead to fact hallucination and error propagation [59].

The emergence of LLM agents has become another promising paradigm for programming tasks. An LLM agent works towards a specified goal by repeatedly interacting with the environment through tools [2, 18, 58, 61, 62]. For example, EnIGMA [2] provides offensive security tools such as decompilers [3] and debuggers for solving Capture The Flag (CTF) challenges. Although EnIGMA has achieved top performance in a public CTF benchmark [42], those contest-level challenges are still far from real-world binary analysis tasks. IDA-Pro-MCP [36] implements an MCP server, which allows LLMs to directly interact with IDA Pro [17] for various RE tasks such as decompilation refinement.

However, we observed that existing binary analysis agents still suffer from two limitations. **First**, their binary code representations are neither analyzer-friendly nor LLM-friendly. EnIGMA [2] only provides two representations: disassembled assembly code and decompiled pseudo-C code. On the one hand, general-purpose LLMs struggle to understand the assembly code well due to its low information density and diversity [23]. On the other hand, decompiled pseudocode may have semantic gaps [12] between the original binary code, which severely hinders sophisticated code analysis techniques. **Second**, LLMs have limited ability to self-reflect [54] without sufficient guidance. When analyzing large binaries, the disassembled and decompiled code for a function could exceed the context limits of LLMs, resulting in the loss of information or conversation history after several rounds of interaction. In consequence, LLMs might get stuck during exploration or even fail to reason about valid next actions [20].

In this paper, we propose a new agentic binary analysis framework ClearAgent, which features a novel binary interface that provides both LLM-friendly and analyzer-friendly tools to facilitate the understanding of binary code semantics and effective vulnerability detection. Specifically, the tools have the following three properties:

Analyzer-friendly. We choose LLVM IR [27] as the main representation as it is closer to the semantics of the binary code than pseudocode [65] and is more structural than assembly code. Additionally, we can perform static analysis algorithms efficiently on the IR and attach rich semantic metadata such as points-to relationships [28] and taint flags [30].

LLM-friendly. We keep the mapping between assembly code, IR, and decompiled pseudo-C code. These mappings enable ClearAgent to perform not only low-level analysis, such as type recovery [55], but also high-level analysis such as function summary [48, 63], and finally propagate analysis results between those representations.

Context-aware. We record the exploration trace in the context, which is represented by functions along the call-graph and basic blocks along the control-flow graph (CFG). ClearAgent can reflect on the context, with an experienced strategy [31, 51], to guide its next action.

Contributions. To summarize, we introduce ClearAgent, a new agentic binary analysis framework that has the following three contributions:

- Friendly interfaces between LLMs and three analyzable formats of binary code: Assembly, IR, and pseudocode;
- Effective context management to guide the agent in exploring the binary for vulnerability locations;
- Preliminary evaluation on the NYU CTF Benchmark and an IoT firmware that demonstrates ClearAgent's promising potential.

2 Design

We first present the overview of ClearAgent in Section 2.1 as an agentic framework. Then we discuss the two core designs in the *Binary Interface*: the *Binary Language Server* in Section 2.2, and the *Context Manager* in Section 2.3. While we emphasize the ability for vulnerability detection, ClearAgent is designed to be a general binary analysis framework for various downstream applications.

2.1 Overview

The overview of ClearAgent is shown in Figure 1. ClearAgent is configured with a system prompt and exploration strategy. The system prompt is an aggregation of the problem description and the interface specification, where the problem description specifies the expected Common Weakness Enumeration (CWE) types and the binary of interest. There are two classes of strategies: the principled strategies, such as DFS versus BFS, backward versus forward [31]; and the top-level strategy, which features a phased design, specifying the principled strategy of each phase. Strategies are prompted using the Chain-of-Thought technique [52].

The LLMs are responsible for understanding the interface's output while supporting function calling. ClearAgent selects general-purpose LLMs (GPT-4 [37] and Claude [4]) as they have demonstrated capabilities to understand different formats of the output, such as assembly code [41], LLVM IR [21], and DOT graph [68].

Binary Database persists all analyzed binaries and serves as the backend of the *Binary Interface*. Once a binary is added, it will first be statically translated to assembly code and LLVM IR [67]. The LLVM IR will be further decompiled to pseudo-C code on demand. Multiple binaries can be loaded in the database for joint analysis across binaries.

Besides the tools defined in the *Binary Interface*, ClearAgent can also call other tools provided by the operating

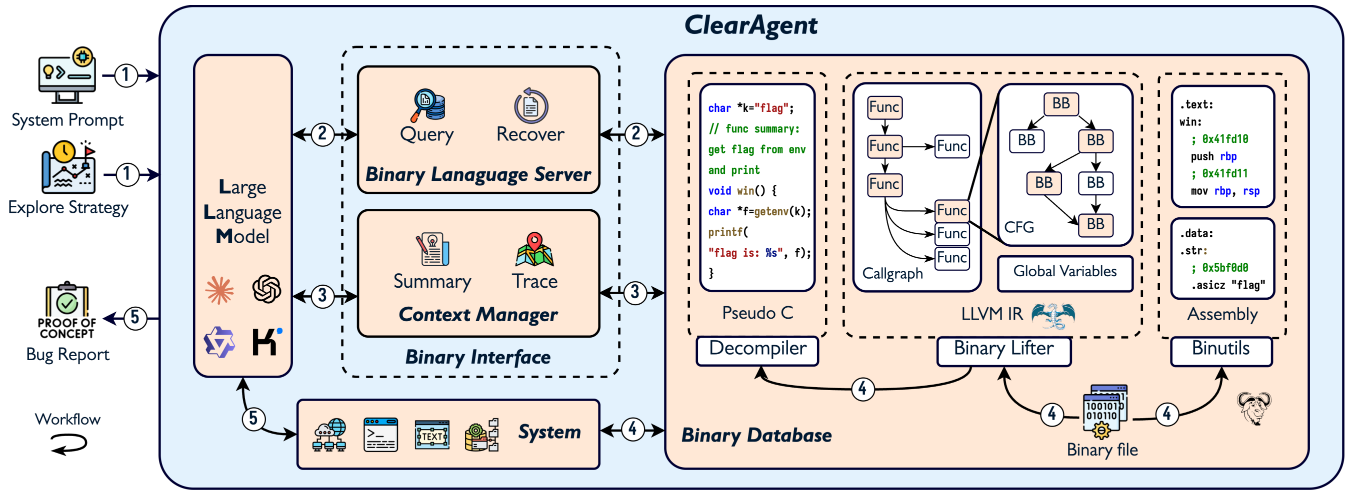

Figure 1. Given system prompt and exploring strategies (①), ClearAgent will start an analysis session. The LLMs work by iteratively calling two sets of tools (②-③) defined in the *Binary Interface*. The *Binary Language Server* is responsible for handling LLMs' function calling (②) with regard to the three analyzable formats of binary code in the *Binary Database*. (④). The server records the summary and trace to the *Context Manager* to guide LLM's next action (③). After rounds of interactions (②-③), the LLMs finally generate verified bug reports with PoC using the system tools (⑤).

system, such as text editing, file system, networking, and terminal. These tools enable ClearAgent to perform auxiliary tasks in binary analysis, including writing Proof-Of-Concept (PoC) scripts and searching keywords in configuration files [9].

2.2 Binary Language Server

The *Binary Language Server* provides two tools via the binary interface: *Query* and *Recover*, with their accepted arguments described in Table 1.

Table 1. Agentic tools defined in the Binary Interface

Tool	Argument	Description
Query	Function	Output the signature, body, summary, and caller/callees
	Basic block	Output the body, summary and predecessors/successors
	Global Var	Output the value, type, references, and strings
Recover	Name/Type	Recover the name/type of functions, local and global vars
Summary	Function/ Basic block	Summarize in natural language with data-flow facts
Trace	Function/ Basic block	Trace the query history of functions/basic blocks

The *Query* tool is primarily used to get information from the LLVM IR. The queried IR objects can be functions, basic blocks, and global variables, which resembles reverse engineers' querying of binary code [16, 64]. The IR objects are organized together with the def-use relationship [24], and the *Query* tool can return not only its own value and type, but also its referenced values. For example, for a function, ClearAgent considers its callers and callees as the next objects to query, which mimics reverse engineers' switching focus to different scopes of the binary [51]. The *Query* tool is also compatible with the other two analyzable formats of binary code. For low-level compatibility, each queried IR object is associated with its address in the binary, enabling the corresponding assembly code/data to be queried on demand. For high-level compatibility, each function is associated with its decompiled pseudocode, which helps the LLMs understand complex control flow more easily.

The *Recover* tool enables ClearAgent to correct or refine the lifted LLVM IR, which mimics the renaming and retyping operations of reverse engineers [47]. The *Recover* tool is used collaboratively with the *Query* tool. For name recovery, the LLMs first query the IR object and recover it with the predicted name. For type recovery, the LLMs will first query the reference sites and assembly code of the IR object, then recover it with the predicted type defined in C syntax. The updated IR objects are further propagated iteratively to all of their reference sites. For example, the recovered variable type could refine the indirect call targets [60].

2.3 Context Management

The *Context Manager* provides two tools via the binary interface: *Summary* and *Trace*, with their accepted arguments described in Table 1.

The *Summary* tool enhances LLMs' memory of all the analyzed binary code. ClearAgent is suggested to summarize the functionality using LLMs when either the function length exceeds the context limit or the function has been analyzed multiple times. The natural language summary is directly given by the LLMs, while the data-flow facts are collected from the symbolic analysis on LLVM IR. Once a function is summarized, the summary will be returned in its next query.

The *Trace* tool features a scoped design for recording the queries. The binary-scope trace records all functions that have been queried, together with their query sequence and calling relationship. The queried functions form subgraph(s) of the binary's call graph as shown in Figure 1. As LLMs cannot accurately identify the control-flow structure [21], ClearAgent further designs a function-scope context to guide intra-procedural exploration. Similarly, the function-scope trace records all queried basic blocks as a subgraph of the current function's CFG. ClearAgent can zoom in/out between binary- and function-scope traces.

The *Trace* tool works collaboratively with the strategy during phased exploration. Phases are predefined by the top-level strategy, such as the analysis phase and verification phase, together with common phases like stuck and finished. Once the exploration gets stuck—namely, performing repetitive queries or exploring existing traces—ClearAgent will be suggested to analyze the trace and either query unexplored functions or enter the next analysis phase.

3 Preliminary Evaluation

We developed two Research Questions (RQs) to evaluate ClearAgent's effectiveness in vulnerability detection. We first evaluate how friendly the binary interface is in helping LLM understand the semantics of binary code (**RQ1**). Then, similar to repo-level exploration [15], we evaluate how effective context management is in helping the ClearAgent explore at the binary level (**RQ2**).

3.1 RQ1: Binary code understanding

RQ1 targets binary challenges from CTF contests, which require collaborative understanding between different formats of binary code. We manually selected **19** out of **90** binary challenges [1] from the Pwn and RE categories in NYU CTF Bench [42] based on two criteria: (1) The challenge targets standard native binary files (e.g., ELF), excluding Python bytecode or raw binary. (2) The challenge is solved by at least one of the SOTA CTF Agents [2, 43].

We acknowledge that the objective of ClearAgent is to find the vulnerability with the PoC script, not to automatically exploit the vulnerability and get the flag. Hence, for Pwn challenges, we consider it a success if ClearAgent can generate a PoC script that reveals the vulnerability. For RE challenges, we require ClearAgent to write a script that directly retrieves the flag. We set up ClearAgent with the Qwen-3 model [57] and the exploration strategy customized for Pwn and RE challenges, respectively.

Table 2. Evaluating ClearAgent performance on binary CTF challenges from the NYU CTF Benchmark [42].

Agent	LLM	Rev	Pwn	All
ClearAgent	Qwen 3 Plus	**7**/11	**8**/8	**15**/19
CRAKEN [43]	Claude 3.5 Sonnet	11/11	6/8	17/19
EnIGMA [2]	Claude 3.5 Sonnet	6/11	6/8	12/19

The overall result is shown in Table 2. ClearAgent successfully understands the binary code semantics and writes PoC scripts in **15** of the CTF challenges. For Pwn challenges, ClearAgent correctly identifies the vulnerable function call and its affected stack variable in LLVM IR, then computes the length to overflow the buffer. Even if the IR variable is lifted as an incorrect type, ClearAgent can still understand the real semantics after querying its corresponding low-level assembly code. As a case study, the 44-byte stack buffer is mistakenly lifted to a 1-byte IR variable in challenge *2023q-pwn-puffin*, while ClearAgent computes the correct payload length of 48 bytes to overflow the critical buffer location. For RE challenges, ClearAgent correctly understands the flag's encoding logic and queries the global variables involved in the computation.

Limitations. We examined the trajectories (reasoning steps) of four failed cases and concluded two main limitations of ClearAgent. We will discuss their solutions in Section 4.

1. **Inaccurate IR.** In *2020f-rev-rap*, ClearAgent fails to disassemble the obfuscated (overlapped) assembly instructions. In *2023q-rev-rebug_2*, ClearAgent writes a PoC with the wrong loop condition and trip count in the lifted IR.
2. **Insufficient Guidance.** In *2021f-rev-maze*, ClearAgent fails to recover the dynamically computed function address at the indirect callsite and stops exploring. In *2019q-rev-gibberish_check*, ClearAgent gets lost in C++ runtime library functions and finally exceeds the rounds limit.

3.2 RQ2: Binary-level exploration

As most of the selected CTF challenges have only simple calling contexts and function logic, the binaries in **RQ1** are not suitable for evaluating ClearAgent's ability under complex calling contexts. Instead, we select a CGI binary from Manta [60]'s IoT firmware dataset as a case study. ClearAgent is prompted with a top-level strategy that contains recovery, analysis, and verification phases.

As shown in Figure 2, during the recovery phase, ClearAgent explores forward with BFS until finding the table initializer and then queries its referenced global variables.

[1] Selected binary list in RQ1: http://bit.ly/4fRVcYV

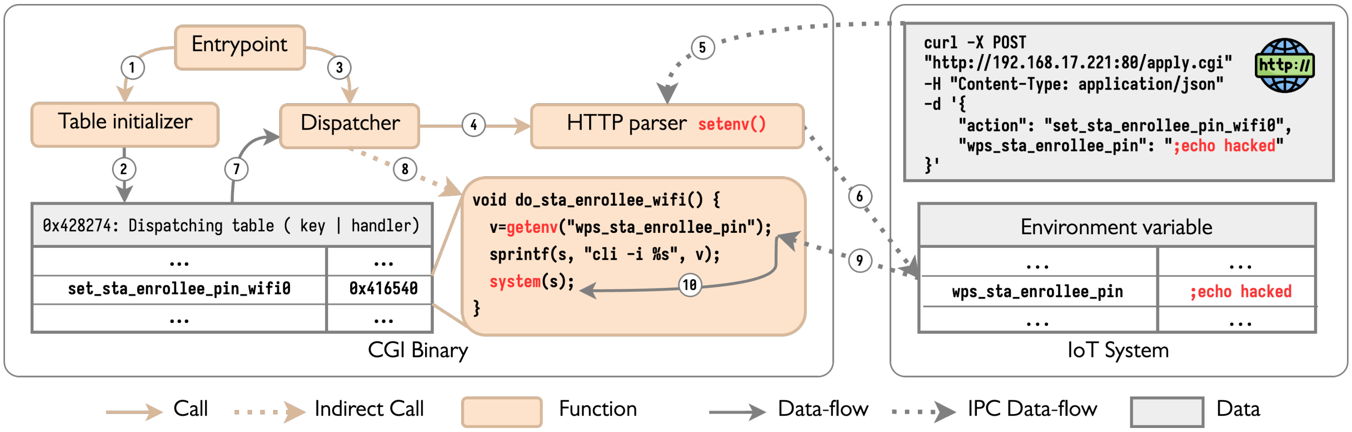

Figure 2. The call relation and data flow in the command injection vulnerability CVE-2022-46593. The CGI binary first initializes (①) the mapping between action string and handler function and stores it in a global table (②). It then calls the dispatcher (③) to listen for HTTP requests from the IoT system (④). Upon request, the parser extracts data (⑤) and stores it in environment variables (⑥). Then the dispatcher selects (⑦) and calls (⑧) the handler function `do_sta_enrollee_wifi` by looking up the `action` field in the dispatching table. Finally, the handler retrieves malicious payload from the environment variable `wps_sta_enrollee_pin` (⑨) and injects it into the system command (⑩).

Since the global variable at address `0x428274` was initially recovered as `char *` in the lifted IR, ClearAgent observes the semantic inconsistency between its type and usage pattern at the dispatcher. ClearAgent then queries data at the address of the global variable, recognizes the array structure, and recovers its type to `struct action[]`. With the recovered table, ClearAgent can understand the mapping between function names and handlers and recover the indirect call targets at the dispatcher.

During the analysis phase, ClearAgent queries and examines each handler function. In the vulnerable handler function at `0x416540`, ClearAgent explores along the def-use chain and identifies the sensitive data flow from `getenv`, via `sprintf`, to `system`. With knowledge of third-party functions [30], ClearAgent records it as a candidate command injection bug. During the verification phase, ClearAgent explores backward with DFS until identifying the environment variable setter, then constructs the payload to match both the handler's function name and the malicious environment variable name.

In comparison, traditional binary analyzers [60] might conservatively scan all functions and identify the buggy data flow inside the handler function, but still require considerable human effort to verify the bug.

4 Future Work

We propose three future directions to address the two limitations discussed in Section 3.1. After that, ClearAgent is expected to be evaluated on larger binary vulnerability datasets [35] and applied to real-world vulnerability finding.

Refinement. ClearAgent should not rely on one-time efforts from traditional analyzers, but aim at refining the binary code on the fly. To address the assembly obfuscation in Limitation 1, ClearAgent can perform superset disassembly [6] on broken assembly code address. To address incorrectly lifted IR in Limitation 1, ClearAgent can utilize instruction-level validation techniques [11] to verify and correct critical IR variables.

Exploration. The strategies need to extend to various analysis scenarios such as multi-binary analysis [39]. In fact, the Limitation 2 also arises in the dilemma of complex library dependencies, where ClearAgent has to decide whether to analyze the library code or search for a similar pattern and speculate its functionality instead. Another practical scenario is huge binaries [32], where ClearAgent can create a program slice that only includes functions of interest to reduce token consumption and analysis overhead. Finally, reviewing ClearAgent trajectories can explain the performance and bring new insights, similar to observing the process of human reverse engineering [31, 51].

Expertise. To improve the performance of each sub-task in vulnerability detection, ClearAgent can integrate tools with stronger semantic reasoning abilities [46], such as constraint solvers to verify the feasibility of vulnerable paths [44], emulation executors [66] to extract code semantics, and specialized LLMs to predict variable names [38, 56] and types [55]. ClearAgent can also integrate a symbolic analysis engine (numerical analysis [34], type-based pointer analysis [14]) to compute candidate addresses at indirect callsites. The symbolic engine can be called on demand when exploring to tackle Limitation 2.

References

[1] Vector 35. 2025. Binary Ninja: An interactive decompiler, disassembler, debugger, and binary analysis platform built by reverse engineers, for reverse engineers. https://binary.ninja/

[2] Talor Abramovich, Meet Udeshi, Minghao Shao, Kilian Lieret, Haoran Xi, Kimberly Milner, Sofija Jancheska, John Yang, Carlos E Jimenez, Farshad Khorrami, Prashanth Krishnamurthy, Brendan Dolan-Gavitt, Muhammad Shafique, Karthik R Narasimhan, Ramesh Karri, and Ofir Press. 2025. EnIGMA: Interactive Tools Substantially Assist LM Agents in Finding Security Vulnerabilities. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=Of3wZhVv1R

[3] National Security Agency. 2025. Ghidra: Software Reverse Engineering Framework. https://ghidra.re/

[4] Anthropic. 2025. Meet Claude, your thinking partner. https://www.anthropic.com/claude

[5] Zion Leonahenahe Basque, Ati Priya Bajaj, Wil Gibbs, Jude O'Kain, Derron Miao, Tiffany Bao, Adam Doupé, Yan Shoshitaishvili, and Ruoyu Wang. 2024. Ahoy SAILR! There is No Need to DREAM of C: A Compiler-Aware Structuring Algorithm for Binary Decompilation. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 361–378. https://www.usenix.org/conference/usenixsecurity24/presentation/basque

[6] Erick Bauman, Zhiqiang Lin, Kevin W Hamlen, et al. 2018. Superset Disassembly: Statically Rewriting x86 Binaries Without Heuristics.. In *NDSS*.

[7] David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J. Schwartz. 2011. BAP: a binary analysis platform. In *Proceedings of the 23rd International Conference on Computer Aided Verification* (Snowbird, UT) *(CAV'11)*. Springer-Verlag, Berlin, Heidelberg, 463–469.

[8] Guoqiang Chen, Huiqi Sun, Daguang Liu, Zhiqi Wang, Qiang Wang, Bin Yin, Lu Liu, and Lingyun Ying. 2025. ReCopilot: Reverse Engineering Copilot in Binary Analysis. arXiv:2505.16366 [cs.CR] https://arxiv.org/abs/2505.16366

[9] Libo Chen, Yanhao Wang, Quanpu Cai, Yunfan Zhan, Hong Hu, Jiaqi Linghu, Qinsheng Hou, Chao Zhang, Haixin Duan, and Zhi Xue. 2021. Sharing More and Checking Less: Leveraging Common Input Keywords to Detect Bugs in Embedded Systems. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 303–319. https://www.usenix.org/conference/usenixsecurity21/presentation/chen-libo

[10] Chris Cummins, Volker Seeker, Dejan Grubisic, Baptiste Roziere, Jonas Gehring, Gabriel Synnaeve, and Hugh Leather. 2025. LLM Compiler: Foundation Language Models for Compiler Optimization. In *Proceedings of the 34th ACM SIGPLAN International Conference on Compiler Construction* (Las Vegas, NV, USA) *(CC '25)*. Association for Computing Machinery, New York, NY, USA, 141–153. doi:10.1145/3708493.3712691

[11] Sandeep Dasgupta, Sushant Dinesh, Deepan Venkatesh, Vikram S. Adve, and Christopher W. Fletcher. 2020. Scalable validation of binary lifters. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 655–671. doi:10.1145/3385412.3385964

[12] Luke Dramko, Jeremy Lacomis, Edward J. Schwartz, Bogdan Vasilescu, and Claire Le Goues. 2024. A Taxonomy of C Decompiler Fidelity Issues. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 379–396. https://www.usenix.org/conference/usenixsecurity24/presentation/dramko

[13] Daniel Engel, Freek Verbeek, and Binoy Ravindran. 2024. On the Decidability of Disassembling Binaries. In *Theoretical Aspects of Software Engineering: 18th International Symposium, TASE 2024, Guiyang, China, July 29 – August 1, 2024, Proceedings* (Guiyang, China). Springer-Verlag, Berlin, Heidelberg, 127–145. doi:10.1007/978-3-031-64626-3_8

[14] Lian Gao and Heng Yin. 2025. BinDSA: Efficient, Precise Binary-Level Pointer Analysis with Context-Sensitive Heap Reconstruction. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA053 (June 2025), 22 pages. doi:10.1145/3728928

[15] Jinyao* Guo, Chengpeng* Wang, Xiangzhe Xu, Zian Su, and Xiangyu Zhang. 2025. RepoAudit: An Autonomous LLM-Agent for Repository-Level Code Auditing. In *Proceedings of the 42nd International Conference on Machine Learning*. *Equal contribution.

[16] HyungSeok Han, JeongOh Kyea, Yonghwi Jin, Jinoh Kang, Brian Pak, and Insu Yun. 2023. QueryX: Symbolic Query on Decompiled Code for Finding Bugs in COTS Binaries . In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 3279–3295. doi:10.1109/SP46215.2023.10179314

[17] Hex-Rays. 2025. IDA Pro: A powerful disassembler, decompiler, and a versatile debugger. https://hex-rays.com/ida-pro

[18] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VtmBAGCN7o

[19] Peiwei Hu, Ruigang Liang, and Kai Chen. 2024. DeGPT: Optimizing Decompiler Output with LLM. In *Network and Distributed System Security (NDSS) Symposium*. San Diego, CA, USA. https://dx.doi.org/10.14722/ndss.2024.24401

[20] Zimo Ji, Daoyuan Wu, Wenyuan Jiang, Pingchuan Ma, Zongjie Li, and Shuai Wang. 2025. Measuring and Augmenting Large Language Models for Solving Capture-the-Flag Challenges. In *CCS*.

[21] Hailong Jiang, Jianfeng Zhu, Yao Wan, Bo Fang, Hongyu Zhang, Ruoming Jin, and Qiang Guan. 2025. Can Large Language Models Understand Intermediate Representations in Compilers?. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=zDieh7VWfN

[22] Linxi Jiang, Xin Jin, and Zhiqiang Lin. 2025. Beyond Classification: Inferring Function Names in Stripped Binaries via Domain Adapted LLMs. In *Network and Distributed System Security (NDSS) Symposium*. San Diego, CA, USA. https://dx.doi.org/10.14722/ndss.2025.240797

[23] Nan Jiang, Chengxiao Wang, Kevin Liu, Xiangzhe Xu, Lin Tan, Xiangyu Zhang, and Petr Babkin. 2025. Nova: Generative Language Models for Assembly Code with Hierarchical Attention and Contrastive Learning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. https://openreview.net/forum?id=4ytRL3HJrq

[24] Richard Johnson and Keshav Pingali. 1993. Dependence-based program analysis. *SIGPLAN Not.* 28, 6 (June 1993), 78–89. doi:10.1145/173262.155098

[25] Omer Katz, Ran El-Yaniv, and Eran Yahav. 2016. Estimating types in binaries using predictive modeling. *SIGPLAN Not.* 51, 1 (Jan. 2016), 313–326. doi:10.1145/2914770.2837674

[26] William Landi. 1992. Undecidability of static analysis. *ACM Lett. Program. Lang. Syst.* 1, 4 (Dec. 1992), 323–337. doi:10.1145/161494.161501

[27] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization* (Palo Alto, California) *(CGO '04)*. IEEE Computer Society, USA, 75.

[28] Chris Lattner, Andrew Lenharth, and Vikram Adve. 2007. Making Context-Sensitive Points-to Analysis with Heap Cloning Practical For The Real World. In *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'07)*. San Diego, California.

[29] Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. 2018. Precision-guided context sensitivity for pointer analysis. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 141 (Oct. 2018), 29 pages. doi:10.

1145/3276511
[30] Puzhuo Liu, Chengnian Sun, Yaowen Zheng, Xuan Feng, Chuan Qin, Yuncheng Wang, Zhenyang Xu, Zhi Li, Peng Di, Yu Jiang, and Limin Sun. 2025. LLM-Powered Static Binary Taint Analysis. *ACM Trans. Softw. Eng. Methodol.* 34, 3, Article 83 (Feb. 2025), 36 pages. doi:10.1145/3711816
[31] Alessandro Mantovani, Simone Aonzo, Yanick Fratantonio, and Davide Balzarotti. 2022. RE-Mind: a First Look Inside the Mind of a Reverse Engineer. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2727–2745. https://www.usenix.org/conference/usenixsecurity22/presentation/mantovani
[32] Xiaozhu Meng, Jonathon M. Anderson, John Mellor-Crummey, Mark W. Krentel, Barton P. Miller, and Srđan Milaković. 2021. Parallel binary code analysis. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Virtual Event, Republic of Korea) *(PPoPP '21)*. Association for Computing Machinery, New York, NY, USA, 76–89. doi:10.1145/3437801.3441604
[33] Nicholas Nethercote and Julian Seward. 2007. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation* (San Diego, California, USA) *(PLDI '07)*. Association for Computing Machinery, New York, NY, USA, 89–100. doi:10.1145/1250734.1250746
[34] Huan Nguyen, Soumyakant Priyadarshan, and R. Sekar. 2024. Scalable, Sound, and Accurate Jump Table Analysis. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 541–552. doi:10.1145/3650212.3680301
[35] Trail of Bits. 2016. DARPA Challenge Binaries on Linux, OS X, and Windows. https://github.com/trailofbits/cb-multios
[36] Duncan Ogilvie. 2025. IDA Pro MCP: Simple MCP Server to allow vibe reversing in IDA Pro. https://github.com/mrexodia/ida-pro-mcp
[37] OpenAI. 2023. GPT-4. https://openai.com/index/gpt-4-research/
[38] Kuntal Kumar Pal, Ati Priya Bajaj, Pratyay Banerjee, Audrey Dutcher, Mutsumi Nakamura, Zion Leonahenahe Basque, Himanshu Gupta, Saurabh Arjun Sawant, Ujjwala Anantheswaran, Yan Shoshitaishvili, Adam Doupé, Chitta Baral, and Ruoyu Wang. 2024. "Len or index or count, anything but v1": Predicting Variable Names in Decompilation Output with Transfer Learning. In *2024 IEEE Symposium on Security and Privacy (SP)*. 4069–4087. doi:10.1109/SP54263.2024.00152
[39] Nilo Redini, Aravind Machiry, Ruoyu Wang, Chad Spensky, Andrea Continella, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. 2020. Karonte: Detecting Insecure Multi-binary Interactions in Embedded Firmware. In *2020 IEEE Symposium on Security and Privacy (SP)*. 1544–1561. doi:10.1109/SP40000.2020.00036
[40] Zihan Sha, Hao Wang, Zeyu Gao, Hui Shu, Bolun Zhang, Ziqing Wang, and Chao Zhang. 2025. llasm: Naming Functions in Binaries by Fusing Encoder-only and Decoder-only LLMs. *ACM Trans. Softw. Eng. Methodol.* 34, 4, Article 93 (April 2025), 22 pages. doi:10.1145/3702988
[41] Xiuwei Shang, Shaoyin Cheng, Guoqiang Chen, Yanming Zhang, Li Hu, Xiao Yu, Gangyang Li, Weiming Zhang, and Nenghai Yu. 2024. How Far Have We Gone in Binary Code Understanding Using Large Language Models . In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–12. doi:10.1109/ICSME58944.2024.00012
[42] Minghao Shao, Sofija Jancheska, Meet Udeshi, Brendan Dolan-Gavitt, haoran xi, Kimberly Milner, Boyuan Chen, Max Yin, Siddharth Garg, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, and Muhammad Shafique. 2024. NYU CTF Bench: A Scalable Open-Source Benchmark Dataset for Evaluating LLMs in Offensive Security. In *Advances in Neural Information Processing Systems*, Vol. 37. 57472–57498. https://proceedings.neurips.cc/paper_files/paper/2024/file/69d97a6493fbf016fff0a751f253ad18-Paper-Datasets_and_Benchmarks_Track.pdf
[43] Minghao Shao, Haoran Xi, Nanda Rani, Meet Udeshi, Venkata Sai Charan Putrevu, Kimberly Milner, Brendan Dolan-Gavitt, Sandeep Kumar Shukla, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, and Muhammad Shafique. 2025. CRAKEN: Cybersecurity LLM Agent with Knowledge-Based Execution. arXiv:2505.17107 [cs.CR] https://arxiv.org/abs/2505.17107
[44] Qingkai Shi, Xiao Xiao, Rongxin Wu, Jinguo Zhou, Gang Fan, and Charles Zhang. 2018. Pinpoint: fast and precise sparse value flow analysis for million lines of code. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Philadelphia, PA, USA) *(PLDI 2018)*. Association for Computing Machinery, New York, NY, USA, 693–706. doi:10.1145/3192366.3192418
[45] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2016. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE Symposium on Security and Privacy*.
[46] Yan Shoshitaishvili, Michael Weissbacher, Lukas Dresel, Christopher Salls, Ruoyu Wang, Christopher Kruegel, and Giovanni Vigna. 2017. Rise of the HaCRS: Augmenting Autonomous Cyber Reasoning Systems with Human Assistance. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 347–362. doi:10.1145/3133956.3134105
[47] Igor Skochinsky. 2021. Igor's tip of the week #42: Renaming and retyping in the decompiler. https://hex-rays.com/blog/igors-tip-of-the-week-42-renaming-and-retyping-in-the-decompiler
[48] Zirui Song, Jiongyi Chen, and Kehuan Zhang. 2024. Bin2Summary: Beyond Function Name Prediction in Stripped Binaries with Functionality-Specific Code Embeddings. *Proc. ACM Softw. Eng.* 1, FSE, Article 3 (July 2024), 23 pages. doi:10.1145/3643729
[49] Hanzhuo Tan, Qi Luo, Jing Li, and Yuqun Zhang. 2024. LLM4Decompile: Decompiling Binary Code with Large Language Models. arXiv:2403.05286 [cs.PL] https://arxiv.org/abs/2403.05286
[50] Freek Verbeek, Nico Naus, and Binoy Ravindran. 2024. Verifiably Correct Lifting of Position-Independent x86-64 Binaries to Symbolized Assembly. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 2786–2798. doi:10.1145/3658644.3690244
[51] Daniel Votipka, Seth Rabin, Kristopher Micinski, Jeffrey S. Foster, and Michelle L. Mazurek. 2020. An Observational Investigation of Reverse Engineers' Processes. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1875–1892. https://www.usenix.org/conference/usenixsecurity20/presentation/votipka-observational
[52] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages.
[53] Wai Kin Wong, Daoyuan Wu, Huaijin Wang, Zongjie Li, Zhibo Liu, Shuai Wang, Qiyi Tang, Sen Nie, and Shi Wu. 2025. DecLLM: LLM-Augmented Recompilable Decompilation for Enabling Programmatic Use of Decompiled Code. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA081 (June 2025), 24 pages. doi:10.1145/3728958
[54] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. Demystifying LLM-Based Software Engineering Agents. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE037 (June 2025), 24 pages. doi:10.1145/3715754
[55] Danning Xie, Zhuo Zhang, Nan Jiang, Xiangzhe Xu, Lin Tan, and Xiangyu Zhang. 2024. ReSym: Harnessing LLMs to Recover Variable and Data Structure Symbols from Stripped Binaries. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and*

Communications Security (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 4554–4568. doi:10.1145/3658644.3670340

[56] Xiangzhe Xu, Zhuo Zhang, Zian Su, Ziyang Huang, Shiwei Feng, Yapeng Ye, Nan Jiang, Danning Xie, Siyuan Cheng, Lin Tan, and Xiangyu Zhang. 2025. GenNM: Generative Neural Machine for Binary Code Decompilation. In *Network and Distributed System Security (NDSS) Symposium*. San Diego, CA, USA. https://dx.doi.org/10.14722/ndss.2025.240276

[57] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 Technical Report. arXiv:2505.09388 [cs.CL] https://arxiv.org/abs/2505.09388

[58] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 50528–50652. https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf

[59] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

[60] Chengfeng Ye, Yuandao Cai, Anshunkang Zhou, Heqing Huang, Hao Ling, and Charles Zhang. 2025. Manta: Hybrid-Sensitive Type Inference Toward Type-Assisted Bug Detection for Stripped Binaries. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4* (Hilton La Jolla Torrey Pines, La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 170–187. doi:10.1145/3622781.3674177

[61] Zheng Yu, Ziyi Guo, Yuhang Wu, Jiahao Yu, Meng Xu, Dongliang Mu, Yan Chen, and Xinyu Xing. 2025. PatchAgent: A Practical Program Repair Agent Mimicking Human Expertise. (2025). https://www.dataisland.org/paper/patchagent.pdf

[62] Zhengmin Yu, Yuan Zhang, Ming Wen, Yinan Nie, Wenhui Zhang, and Min Yang. 2025. CXXCrafter: An LLM-Based Agent for Automated C/C++ Open Source Software Building. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE116 (June 2025), 23 pages. doi:10.1145/3729386

[63] Bolun Zhang, Zeyu Gao, Hao Wang, Yuxin Cui, Siliang Qin, Chao Zhang, Kai Chen, and Beibei Zhao. 2025. BinQuery: A Novel Framework for Natural Language-Based Binary Code Retrieval. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA052 (June 2025), 23 pages. doi:10.1145/3728927

[64] Haiquan Zhang. 2023. CodeQL: Also a Powerful Binary Analysis Engine. https://i.blackhat.com/BH-US-23/Presentations/US-23-Zhang-CodeQL-Also-a-Powerful-Binary-Analysis-Engine.pdf

[65] Jianzhou Zhao, Santosh Nagarakatte, Milo M.K. Martin, and Steve Zdancewic. 2012. Formalizing the LLVM intermediate representation for verified program transformations. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Philadelphia, PA, USA) *(POPL '12)*. Association for Computing Machinery, New York, NY, USA, 427–440. doi:10.1145/2103656.2103709

[66] Anshunkang Zhou, Yikun Hu, Xiangzhe Xu, and Charles Zhang. 2024. ARCTURUS: Full Coverage Binary Similarity Analysis with Reachability-guided Emulation. *ACM Trans. Softw. Eng. Methodol.* 33, 4, Article 96 (April 2024), 31 pages. doi:10.1145/3640337

[67] Anshunkang Zhou, Chengfeng Ye, Heqing Huang, Yuandao Cai, and Charles Zhang. 2024. Plankton: Reconciling Binary Code and Debug Information. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 912–928. doi:10.1145/3620665.3640382

[68] Bocheng Zou, Mu Cai, Jianrui Zhang, and Yong Jae Lee. 2024. VGBench: Evaluating Large Language Models on Vector Graphics Understanding and Generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 3647–3659. https://aclanthology.org/2024.emnlp-main.213

Received 2025-07-06; accepted 2025-08-08