Research Statement

Penghui Li

My research goal is to *make software systems secure and reliable*. Software underpins critical infrastructure in every aspect of modern life, yet modern systems have grown immensely in both scale and heterogeneity, often comprising millions of lines of code written in multiple languages. Sophisticated adversaries routinely exploit vulnerabilities in these systems, leading to service disruptions and economic losses. Securing such complex systems requires *scalability* to analyze large codebases, *rigor* to ensure precise reasoning about vulnerabilities, and *adaptability* to keep pace with evolving threats.

My research vision is to realize *autonomous agentic security analysis* that unifies scalability, rigor, and adaptability in a single framework. My past work has advanced foundational security analysis to achieve scalability and rigor through language-agnostic vulnerability detection and execution-efficient validation techniques. However, like most conventional approaches, these techniques still require substantial manual effort from security experts to encode detection logic and adapt to new threat models. Meanwhile, the emerging trend of LLM-based code analysis offers adaptability through natural language prompts that reduce manual specification effort, yet lacks the rigor and scalability required for dependable security reasoning. To bridge this gap, I develop agentic program analysis, where intelligent agents autonomously orchestrate specialized analysis components to discover and validate complex threats. For example, I developed a system that detects privilege escalation in polyglot microservices by unifying program analysis and LLM reasoning over a language-agnostic cross-service program representation. Looking forward, I will continue advancing this approach to secure emerging software paradigms, discover novel threat classes, and enable autonomous defense that continuously evolves without expert intervention.

My work has made significant real-world impacts. I have applied my approaches to securing web applications [1, 2, 3, 10, 4], cloud software [8], desktop applications [5, 6, 11, 7], operating systems [12, 13], and database systems [14]. My research has been published at top-tier venues in security (S&P, Security, CCS, NDSS) and software engineering (ICSE, FSE, ASE), including seven papers as the first author. I have received a Distinguished Paper Award at CCS 2024 [2], a Best Paper Honorable Mention at CCS 2022 [10], and a Distinguished Artifact Award at CCS 2025 [15]. My work has uncovered 346 previously unknown vulnerabilities in widely deployed software, including the Linux kernel, the PHP interpreter, and GitHub, resulting in 47 assigned CVEs. These discoveries were acknowledged by vendors, rewarded through bug bounty programs, and patched to protect millions of users worldwide.

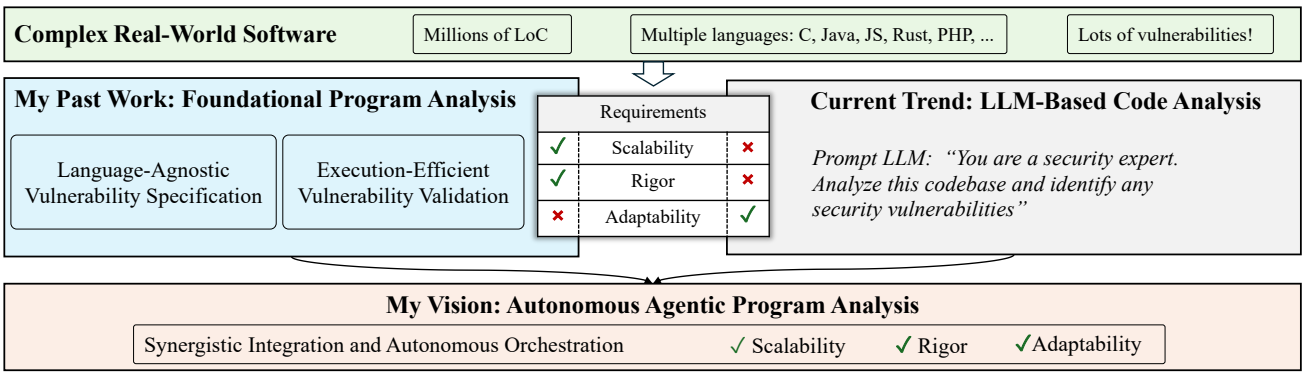

Figure 1: An overview of my past and current research.

Foundational Program Analysis

To achieve scalability and rigor in vulnerability analysis for complex real-world software comprising millions of lines across multiple languages, I develop a domain-specific language (DSL)-driven framework for systematic detection and execution-efficient dynamic testing for concrete confirmation.

Language-Agnostic Vulnerability Specification. To identify vulnerable code components in software systems, I formulate this task as a *code search problem* that retrieves program elements satisfying specific criteria. My key

innovation is to separate the representation of the program from the search criteria, enabling language-agnostic analysis. Specifically, I abstract language-specific complexities into a code database and develop a DSL framework for expressing vulnerability specifications as simple, declarative queries. Unlike prior approaches that require complex graph query languages [17], my DSL distills vulnerability detection into four elementary operation types (name lookup, operation lookup, call graph traversal, and data-flow tracking). Each operation corresponds to a fundamental program analysis task, and these four operations cover over 90% of common analysis tasks. Complex vulnerability patterns can be systematically specified by composing these elementary operations. This general framework enables scalable static detection across heterogeneous codebases while alleviating the need for security experts to develop language-specific detection logic from scratch.

I demonstrated the expressiveness and generality of this approach by detecting diverse vulnerability classes across multiple language paradigms. The approach uncovered *over 60 previously unknown vulnerabilities, including 25 CVEs*, across widely-deployed systems [3, 10]. Notably, it revealed a fundamental design flaw in PHP's type system that influenced the language design of PHP 8.0 [3].

Execution-Efficient Vulnerability Validation. I rigorously validate potential vulnerabilities through execution-efficient dynamic testing that confirms exploitability with concrete runtime evidence. Many prior solutions focus on improving algorithmic strategies for path exploration [6, 7, 16]. However, my systematic profiling of real-world web applications revealed a different bottleneck where execution efficiency is instead the limiting factor. Specifically, accesses to external resources such as databases and network services dominate execution time (around 50%), as each test iteration independently re-fetches the same data, and even small slowdowns compound exponentially across thousands of repeated executions. This insight fundamentally shifts the optimization target from path selection to execution speed. Based on this insight, I developed a novel software-based data caching mechanism that transparently intercepts and stores frequently accessed external data in shared memory, and a just-in-time code compilation technique that optimizes interpreted language execution [2]. This solution is generic and applicable to any dynamic analysis tool as a plug-in component, requiring no modifications to the target application or analysis tool. By shifting the optimization focus from algorithmic strategies to execution efficiency, this approach enables dynamic testing to scale to real-world applications that were previously too slow to analyze effectively.

My execution-efficient validation techniques improve throughput by *up to 4*$\times$ and expose vulnerabilities *105%* more quickly. Applying these techniques, I discovered new vulnerabilities in WordPress, which powers nearly half of all websites worldwide [2].

Autonomous Agentic Program Analysis

Building on the scalability and rigor of foundational techniques, I develop autonomous agentic program analysis that integrates LLM reasoning with rigorous program analysis to achieve adaptability.

Synergistic Integration and Autonomous Orchestration. I develop an agentic architecture where intelligent agents autonomously orchestrate specialized analysis components, dynamically decompose complex analysis tasks into adaptive sequences, and chain results across heterogeneous systems to reconstruct complete attack paths. This orchestration integrates two complementary capabilities. DSL-based program analysis provides systematic code search across heterogeneous codebases, performs precise data flow tracing through call chains, validates LLM-generated hypotheses against actual code, and discovers concrete attack paths [8, 9]. Meanwhile, LLMs interpret implicit security requirements from natural language documentation and API specifications, formulate analysis strategies based on high-level security policies, and reason about semantic properties across system boundaries. The two components work synergistically to validate each other: when an LLM infers security requirements from documentation, program analysis confirms this by tracing code execution to verify whether corresponding security checks exist; conversely, when program analysis identifies data flows crossing boundaries, LLM reasoning interprets the security implications by analyzing related policies and documentation. This eliminates the need for security experts to manually encode detection logic or adapt to evolving threat models.

I applied this agentic approach to analyze polyglot microservice systems in production cloud environments. The

unified cross-service abstraction enables seamless analysis across multiple languages and automatic reconstruction of end-to-end attack paths spanning service boundaries. It discovered *20 critical privilege escalation vulnerabilities* in widely-used cloud applications that existing program analysis tools and LLMs alone could not detect.

Future Directions

My future research will advance my vision of autonomous agentic program analysis through three interconnected thrusts. Grounded in the core principles of synergistic integration and autonomous orchestration, these thrusts form a unified path toward fully autonomous security by first understanding emerging behaviors, then discovering novel threats, and finally creating autonomous defense

Securing Emerging Software Paradigms. My first research thrust will address the fundamentally new security challenges introduced by rapidly evolving software paradigms. I am particularly interested in securing *agentic software systems*, where traditional deterministic code interacts with non-deterministic LLM-driven agents that make decisions and take actions. These systems exhibit emergent behaviors that create subtle, cross-component security risks invisible to traditional analysis. For instance, cooperating agents may inadvertently bypass authorization checks or chain API calls into unsafe states unreachable by individual calls. I will first engage hands-on with real agentic systems to identify critical threat models, then develop practical analysis solutions. I will develop abstractions that capture both *control-flow* (agent decisions and actions) and *data-flow* (information propagation across agents) to systematically model software behaviors, and decompose the analysis into manageable components that can be addressed by combining program analysis with LLM reasoning. This line of research aims to establish foundational techniques for reasoning about security in agentic systems, including detection of unsafe state compositions, validation of information flow in probabilistic environments, and verification of policies across agent boundaries. Beyond security, I will explore the privacy and safety of agentic software systems.

Discovering Novel Threat Classes. Current security techniques largely detect *new instances* of known vulnerability classes by relying on predefined vulnerability patterns. My goal is to develop methods that autonomously discover *new categories* of vulnerabilities and rigorously demonstrate their exploitability. My recent work [9] demonstrates how LLMs move beyond manually specified patterns to automatically generate detection patterns. The preliminary results show the feasibility of discovering effective detection patterns that are often overlooked by human experts. Building on this, I will develop methods that combine program analysis with learning-based reasoning to identify novel threat classes. I will first study how human security experts discover new threat classes, specifically their process of reasoning about program semantics, anticipating corner cases, and extrapolating from past failures. I will then systematically replicate this capability in automated systems by decomposing the discovery process into components that start from fundamental security properties (*e.g.*, confidentiality, integrity, availability), generate hypotheses about potential violations, explore execution paths to trigger them, and validate exploitability through concrete demonstrations. For example, the system might discover privilege escalation patterns by chaining multiple legitimate operations that are individually authorized but collectively achieve unauthorized access. This direction will transform software security from pattern matching to proactive threat discovery.

Creating Real-Time Defense. Building on the capability to discover novel threats, my future research will enable software to autonomously synthesize, validate, and deploy defenses in real time. My recent work on runtime protection [13] demonstrated automated quarantine of malicious inputs to block exploitation. I will build systems that infer security policies from vulnerability evidence and formally verify their correctness to guarantee they block attacks without disrupting legitimate functionality. I will develop automated verification techniques that efficiently check policy correctness against both security specifications and functional requirements, enabling safe deployment in production systems. Beyond individual vulnerabilities, the system will learn from attack patterns to preemptively strengthen defenses and predict future threat vectors. I will investigate real defense deployment workflows to identify critical bottlenecks, then decompose autonomous defense into manageable components solved by combining program analysis, formal verification, and LLM reasoning. This research will create self-healing software systems that continuously evolve their defenses to provide proactive, autonomous protection.

References

[1] Penghui Li, Wei Meng, Mingxue Zhang, Chenlin Wang, and Changhua Luo. "Holistic Concolic Execution for Dynamic Web Applications via Symbolic Interpreter Analysis". In *Proceedings of the 45th IEEE Symposium on Security and Privacy (S&P)*. May 2024.

[2] Penghui Li and Mingxue Zhang. "FuzzCache: Optimizing Web Application Fuzzing Through Software-Based Data Cache". In *Proceedings of the 31st ACM Conference on Computer and Communications Security (CCS)*. Oct. 2024. Distinguished Paper Award.

[3] Penghui Li and Wei Meng. "LChecker: Detecting Loose Comparison Bugs in PHP". In *Proceedings of the Web Conference (WWW)*. Apr. 2021.

[4] Penghui Li, Wei Meng, Kangjie Lu, and Changhua Luo. "On the Feasibility of Automated Built-in Function Modeling for PHP Symbolic Execution". In *Proceedings of the Web Conference (WWW)*. Apr. 2021.

[5] Penghui Li, Wei Meng, and Kangjie Lu. "SEDiff: Scope-Aware Differential Fuzzing to Test Internal Function Models in Symbolic Execution". In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. Nov. 2022.

[6] Penghui Li, Wei Meng, and Chao Zhang. "SDFuzz: Target States Driven Directed Fuzzing". In *Proceedings of the 33rd USENIX Security Symposium (Security)*. Aug. 2024.

[7] Penghui Li, Yinxi Liu, and Wei Meng. "Understanding and Detecting Performance Bugs in Markdown Compilers". In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Nov. 2021. Best Software Artifact Nomination.

[8] Penghui Li, Hong Yau Chong, Yinzhi Cao, and Junfeng Yang. "Detecting Privilege Escalation in Polyglot Microservices via Agentic Program Analysis". Under Review.

[9] Penghui Li, Songchen Yao, Josef Sarfati Korich, Changhua Luo, Jianjia Yu, Yinzhi Cao, and Junfeng Yang. "Automated Static Vulnerability Detection via a Holistic Neuro-Symbolic Approach". Under Review, `https://arxiv.org/abs/2504.16057`.

[10] Changhua Luo, Penghui Li, and Wei Meng. "TChecker: Precise Static Inter-Procedural Analysis for Detecting Taint-Style Vulnerabilities in PHP Applications". In *Proceedings of the 29th ACM Conference on Computer and Communications Security (CCS)*. Nov. 2022. Best Paper Honorable Mention.

[11] Jiayi Lin, Changhua Luo, Mingxue Zhang, Lanteng Lin, Penghui Li, and Chenxiong Qian. "Fuzzing JavaScript Engines by Fusing JavaScript and WebAssembly". In *Proceedings of the 48th International Conference on Software Engineering (ICSE)*. Apr. 2026.

[12] Ming Yuan, Bodong Zhao, Penghui Li, Jiashuo Liang, Xinhui Han, Xiapu Luo, and Chao Zhang. "DDRace: Finding Concurrency UAF Vulnerabilities in Linux Drivers with Directed Fuzzing". In *Proceedings of the 32nd USENIX Security Symposium (Security)*. Aug. 2023.

[13] Yuan Li, Chao Zhang, Jinhao Zhu, Penghui Li, Chenyang Li, Songtao Yang, and Wende Tan. "VulShield: Protecting Vulnerable Code Before Deploying Patches". In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*. Feb. 2025.

[14] Zeyang Zhuang, Penghui Li, Pingchuan Ma, Wei Meng, and Shuai Wang. "Testing Graph Database Systems via Graph-Aware Metamorphic Relations". In *Proceedings of the 50th International Conference on Very Large Data Bases (VLDB)*. Aug. 2024.

[15] Andreas D. Kellas, Neophytos Christou, Wenxin Jiang, Penghui Li, Laurent Simon, Yaniv David, Vasileios P. Kemerlis, James C. Davis, and Junfeng Yang. "PickleBall: Secure Deserialization of Pickle-Based Machine Learning Models". In *Proceedings of the 32nd ACM Conference on Computer and Communications Security (CCS)*. Oct. 2025. Distinguished Artifact Award.

[16] Changhua Luo, Wei Meng, and Penghui Li. "SelectFuzz: Efficient Directed Fuzzing with Selective Path Exploration". In *Proceedings of the 44th IEEE Symposium on Security and Privacy (S&P)*. May 2023.

[17] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. "Modeling and Discovering Vulnerabilities with Code Property Graphs". In *Proceedings of the 35th IEEE Symposium on Security and Privacy (S&P)*. May 2014.