Research Statement

Penghui Li

My research goal is to *make software systems secure and reliable*. Software underpins critical infrastructure in everyday life, yet modern systems have grown immensely complex in both scale and heterogeneity, often comprising millions of lines of code written in multiple languages. This complexity creates an expanding attack surface that sophisticated adversaries routinely exploit, leading to system compromises, service disruptions, and significant economic losses. Securing such complex software 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 unify scalability, rigor, and adaptability in a holistic security analysis framework.*

My research philosophy toward this vision is *principled abstraction and decomposition*. Abstraction eliminates irrelevant complexity to reveal essential security properties, while decomposition breaks intractable challenges into solvable sub-problems. By choosing appropriate abstraction layers and decomposition granularities, my research identifies critical bottlenecks in security practices and develops deployable defenses with real-world impact. Guided by this philosophy, my past work has advanced foundational security analysis to achieve scalability and rigor through language-agnostic vulnerability detection and execution-efficient validation. However, like most conventional approaches, these techniques still require substantial manual effort from security experts to design detection logic and adapt to new threat models. Meanwhile, the emerging trend of LLM-based code analysis offers adaptability through natural language prompts, yet lacks the computational scalability and logical rigor required for dependable security reasoning. To bridge this gap, I have been developing *agentic program analysis*, where intelligent agents coordinate specialized analysis components to exploit their complementary strengths. Specifically, an LLM agent can infer security intent from documentation, while program analysis can validate the code implementation.

My work has made significant real-world impacts. I have applied my philosophy to securing web applications [2, 4, 6, 7, 12], desktop applications [1, 3, 5], cloud software [8], operating systems [11, 14], and database systems [15]. 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 [7], a Best Paper Honorable Mention at CCS 2022 [12], and a Distinguished Artifact Award at CCS 2025 [10]. 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.

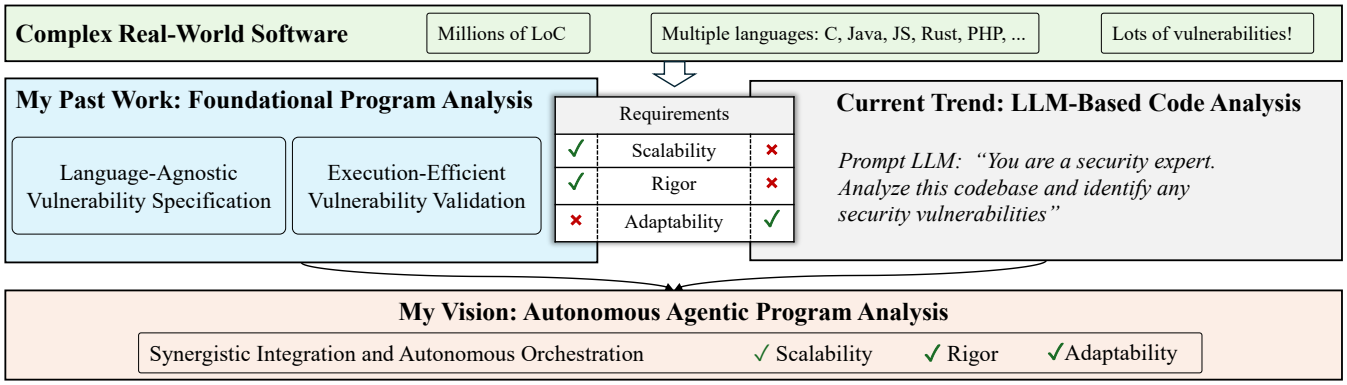

Figure 1: An overview of my past and current research.

Foundational Program Analysis

To achieve scalability and rigor in vulnerability analysis for complex real-world software, I develop a domain-specific language (DSL) driven framework for systematic detection and execution-efficient dynamic testing for validation. These two foundational techniques have a broad impact across software security and also serve as the foundation for agentic program analysis.

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 [16], 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 [2, 12]. Notably, it revealed a fundamental design flaw in PHP's type system that influenced the language of PHP 8.0 [2].

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 [1, 5, 13]. However, my systematic profiling of real-world web applications revealed the true bottleneck is execution efficiency rather than path exploration strategies. Specifically, accesses to external resources such as databases and network services dominate execution time, 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 [7]. 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. This approach dramatically improves validation throughput by *up to 4*$\times$ and exposes vulnerabilities *2*$\times$ faster. It enabled the discovery of critical zero-day vulnerabilities in WordPress that threatened nearly half of the global web and would have remained undetected with prior approaches [7].

Current Work Toward Vision: Autonomous Agentic Program Analysis

To achieve adaptability while maintaining scalability and rigor, I develop agentic program analysis that synergistically integrates LLM reasoning with foundational program analysis techniques.

Synergistic Integration and Autonomous Orchestration. I design an agentic architecture where intelligent agents coordinate 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-driven program analysis provides systematic code search and performs precise data flow tracing through call chains [8, 9]. Meanwhile, LLMs interpret implicit security requirements from natural language documentation, formulate analysis strategies based on high-level security policies, and reason about semantic properties across system boundaries. Critically, LLMs compose sequences of DSL elementary operations by determining which to invoke, chaining them based on intermediate results, and adapting strategies as new evidence emerges. For example, to detect privilege escalation across microservices, the system traces data flows across service boundaries while validating authorization policies at each step. When an LLM infers security requirements from documentation, program analysis verifies whether corresponding checks exist in the code; conversely, when program analysis identifies suspicious data flows, LLM reasoning determines their security implications. This synergy achieves *scalability* through automated orchestration across large codebases, *rigor* through program analysis validation of LLM hypotheses, and *adaptability* through security requirement specification.

I applied this agentic approach to analyze polyglot microservice systems in production cloud environments. The unified cross-service abstraction enables seamless analysis across programming languages (C, Java, JavaScript, Rust,

PHP, *etc.*) and automatic reconstruction of end-to-end attack paths spanning service boundaries. It discovered *24 critical privilege escalation vulnerabilities* in widely-used cloud applications that existing program analysis tools and LLMs alone could not detect [8]. It also demonstrated high usability and extensibility to other software domains and vulnerability categories.

Future Directions

My future research will advance my vision of autonomous agentic program analysis through three interconnected thrusts. Building on 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 dynamic 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.

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 automatically 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 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 addresses the fundamental limitation that existing techniques can only discover instances of predefined vulnerability classes.

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 [11] 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, 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.

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

[3] 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.

[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 Chao Zhang. "SDFuzz: Target States Driven Directed Fuzzing". In *Proceedings of the 33rd USENIX Security Symposium (Security)*. Aug. 2024.

[6] 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.

[7] 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**.

[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] 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**.

[11] 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.

[12] 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**.

[13] 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.

[14] 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.

[15] 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.

[16] 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.