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—properties that existing approaches fail to achieve simultaneously. *My research vision is to unify scalability, rigor, and adaptability for security analysis*.

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, I advanced foundational vulnerability analysis to achieve scalability and rigor through abstractions of programming language features. However, like most classic 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 it lacks computational scalability and logical rigor. To bridge this gap, I have been developing *agentic program analysis*, where an intelligent agent coordinates LLM reasoning with classic program analysis to exploit their complementary strengths. By abstracting the interaction interfaces between LLMs and large codebases through structured analysis primitives, this approach enables LLMs to provide adaptability while ensuring scalability and rigor.

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, 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 [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.

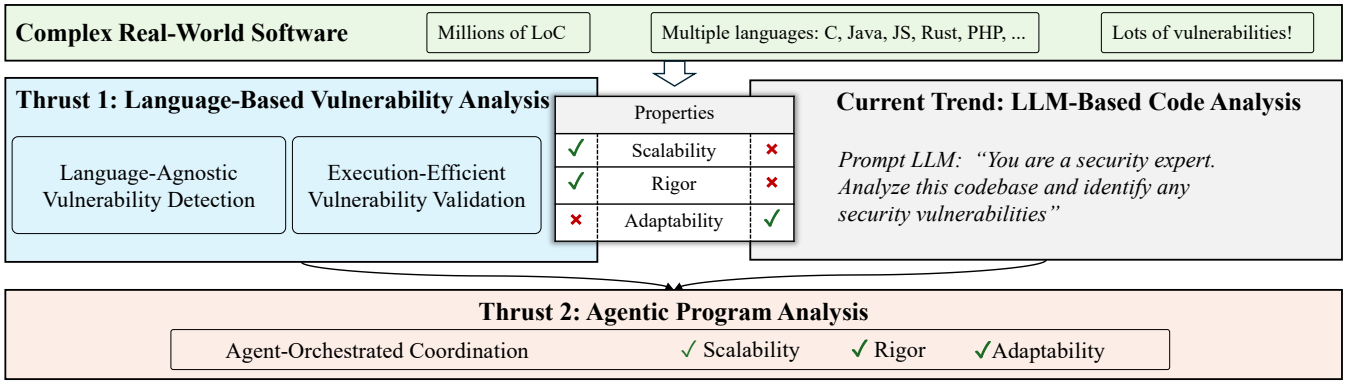

Figure 1: An overview of my past and current research.

Thrust 1: Language-Based Vulnerability Analysis

Modern software systems are polyglot, implemented in multiple programming languages. The use of diverse language features introduces significant challenges for security analysis. Meanwhile, even subtle misuse of language features leads to security vulnerabilities. For example, the type system in dynamic languages is difficult to analyze

and can lead to type juggling attacks [2]. Prior analysis tools require substantial engineering effort to support even a single language and cannot scale to polyglot systems. I develop new vulnerability detection and validation techniques to achieve scalability and rigor through language-based analysis.

Language-Agnostic Vulnerability Detection. My key insight is to formulate vulnerability detection as a *code search problem* that retrieves program elements satisfying specific criteria. By separating the representation of programs from the search criteria, this abstraction enables language-agnostic analysis. Specifically, I abstract language-specific complexities into a unified code database and develop a domain-specific language (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 code search primitives: name lookup, abstract-syntax-tree lookup, flow tracking, and call graph traversal. Each primitive corresponds to a fundamental program analysis task, and these four primitives cover over 90% of common analysis tasks. Complex vulnerability patterns can thus be systematically specified by composing these elementary primitives. 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. 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 design of PHP 8.0 [2].

Execution-Efficient Vulnerability Validation. Dynamic testing validates vulnerabilities by executing programs with concrete inputs to confirm exploitability. However, this process is prohibitively slow for real-world applications due to the overhead of executing complex language features. My key insight is that not all language feature executions are necessary for validation. By abstracting away irrelevant execution overhead, I can dramatically improve efficiency without sacrificing correctness. My systematic profiling of real-world web applications revealed that accessing external resources through language-specific APIs, such as database queries and network calls, dominates execution time. Each test iteration independently re-fetches the same external resources, and even small slowdowns compound exponentially across thousands of repeated executions. 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, combined with a just-in-time code compilation technique that optimizes interpreted language execution [7]. 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].

Thrust 2: Agentic Program Analysis

Classic language-based analysis achieves scalability and rigor but can still take expert weeks or months to design detection logic for each vulnerability pattern. LLM-based code analysis offers adaptability through natural language prompts but lacks the computational scalability and logical rigor. For instance, prior LLM-based code agents directly feed entire codebases into LLMs for security assessment [15], which is constrained by context window limits and prone to hallucinations. To achieve adaptability while maintaining both scalability and rigor, I develop agentic program analysis that effectively coordinates LLM reasoning with program analysis.

Agent-Orchestrated Coordination. My approach applies principled decomposition to bridge LLMs and program analysis as complementary reasoning systems. LLMs excel at interpreting implicit security requirements from natural language cues (*e.g.*, comments) and formulating high-level analysis strategies, while program analysis provides systematic code search and rigorous validation. Specifically, I decompose security analysis into elementary components that are handled by either LLMs for semantic reasoning or program analysis through DSL primitives for code examination. LLMs act as orchestrators that compose these components into analysis plans, determining when to apply respective components [8, 9]. For example, an LLM might identify sinks for taint-style vulnerabilities through semantic reasoning, then compose DSL primitives to trace data flows through the code, and finally interpret the trace results to validate security. These DSL primitives enable on-demand context retrieval, selectively fetching relevant code snippets rather than processing entire codebases, and provide concrete execution evidence to validate LLM-generated hypotheses. This division of labor achieves *adaptability* through LLM-driven strategy formulation,

scalability through efficient on-demand retrieval, and *rigor* through program analysis validation.

I applied this approach to detect privilege escalation vulnerabilities in complex microservice systems. An LLM infers from service documentation that certain operations require administrator privileges, then composes DSL primitives to trace how user inputs flow to these operations. Program analysis executes these traces across multiple services and validates whether proper authorization checks exist at each step. This coordinated analysis discovered *24 critical privilege escalation vulnerabilities* in widely-used cloud applications that neither standalone program analysis tools nor LLMs could detect [8].

Future Directions

My past research has laid a solid program analysis foundation for software systems today. Next-generation software is evolving with non-deterministic AI components, heterogeneous architectures, and increasingly sophisticated attacks, which all demand new analytical capabilities beyond what exists today. My future research will continue applying my research philosophy to develop new security foundations that analyze emerging software paradigms, unlock new security capabilities, and enable perpetual security evolution.

Securing Next-Generation Software. Software is evolving beyond traditional deterministic paradigms to incorporate AI-driven decision-making, autonomous agents, and probabilistic reasoning. These systems make non-deterministic decisions, adapt at runtime, and exhibit emergent behaviors that traditional security analysis cannot adequately reason about. For example, agentic software systems where LLM-driven agents interact with traditional code create subtle cross-component security risks invisible to conventional analysis. My research will develop unified abstractions that enable security analysis across diverse software paradigms by capturing control-flow and data-flow in heterogeneous, non-deterministic systems. I will establish foundational techniques applicable to any emerging paradigm, including detection of unsafe emergent states, validation of information flow in probabilistic environments, and verification of policies across heterogeneous system boundaries.

Enabling New Security Capabilities. Beyond scalability, rigor, and adaptability, security analysis requires new capabilities such as accessibility for practical adoption, resilience for continuous operation, and provability for formal guarantees. My research philosophy of principled decomposition and composition will provide the foundation to achieve these by enabling independent reasoning about each component and automatically composing component-level properties into system-wide guarantees without re-analyzing the entire system. For example, in cloud platforms with thousands of microservices, developers could verify each service once and automatically derive end-to-end security guarantees, with incremental updates only when specific services change. My research will develop compositional abstractions that capture security properties at component boundaries, formal frameworks that preserve properties under composition, and incremental algorithms that maintain guarantees as systems evolve. These foundations will unlock previously impossible capabilities such as planetary-scale analysis, continuous assurance for evolving systems, and provable security guarantees across entire software ecosystems.

Achieving Perpetual Security Evolution. The ultimate vision is security systems that evolve autonomously, discovering emerging vulnerability classes, synthesizing verified defenses, and continuously improving without human intervention. Current security practices still require excessive expertise and engineering efforts for each of these essential components. My research will establish a new paradigm where security systems evolve autonomously across the *entire security lifecycle*, from discovering vulnerabilities and understanding attack patterns to generating defenses and deploying protections through continuous learning and adaptation. This requires learning from past vulnerabilities to predict future attack surfaces, reasoning about novel threat compositions, autonomously generating defense policies with formal guarantees, and evolving analysis strategies based on accumulated experience. I will develop foundations that enable autonomous evolution through composable reasoning components, establish formal frameworks for verifying autonomously generated defenses, and create learning mechanisms that improve while preserving soundness. For example, systems might discover vulnerability principles from attack patterns, generate and verify defense policies, and learn from outcomes to refine future reasoning. This will fundamentally transform security from reactive defense to autonomous evolution, enabling self-improving systems and establishing foundations for trustworthy autonomous intelligence across all critical domains.

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] 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.

[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] 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. "EnIGMA: Interactive Tools Substantially Assist LM Agents in Finding Security Vulnerabilities". In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. June 2025.

[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.