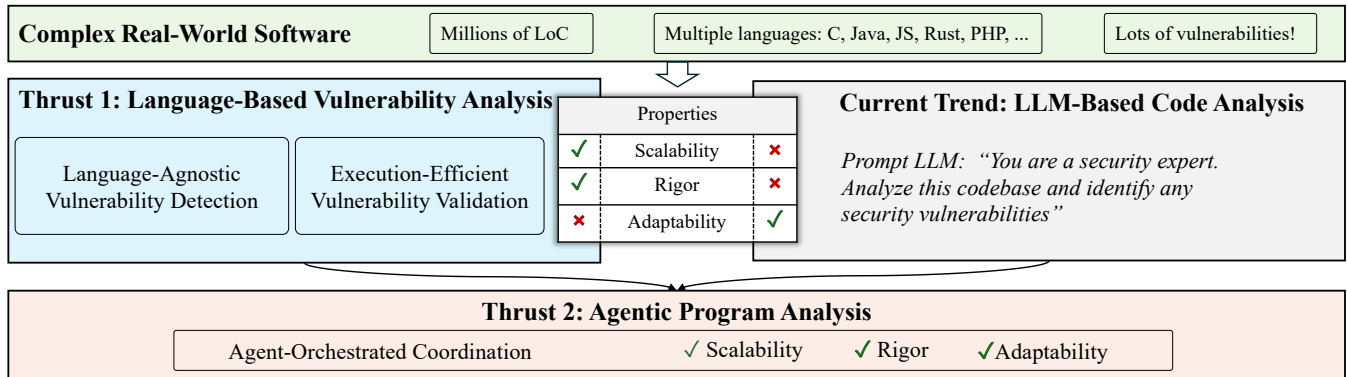# Research Statement

Penghui Li

My research interest lies at the intersection of security, software engineering, and machine learning. I focus on developing automated vulnerability analysis to detect and prevent critical flaws before they are exploited. Specifically, modern systems have grown immensely complex, introducing security flaws that carry severe real-world consequences. Effective vulnerability analysis requires *scalability* to handle these large codebases, *rigor* to ensure precise reasoning about program semantics, and *adaptability* to keep pace with evolving threats. However, despite decades of progress, current practices fail to achieve these properties simultaneously. They rely on security experts to model programming language semantics, develop vulnerability patterns, and encode heuristic rules into automated analysis tools. This approach is costly to build, brittle to software changes, and rarely generalizes across code. **My research vision is to unify scalability, rigor, and adaptability for vulnerability 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 **language-based vulnerability analysis** to achieve scalability and rigor through abstractions of programming language features. However, like most classic approaches, these techniques still require security experts to manually 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.

Research Impact. Guided by this philosophy, my work has achieved significant real-world impact across diverse domains. I have successfully secured systems ranging from web applications [2, 4, 6, 7, 12], desktop applications [1, 3, 5], and cloud software [8] to operating systems [11, 13] and databases [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.

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]. 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 and led to a patch in the core PHP interpreter [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×* and exposes vulnerabilities *2×* 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

While classic language-based analysis achieves scalability and rigor, it 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 foundation for analyzing today's complex software systems. However, future software is evolving with non-deterministic AI components, heterogeneous architectures, and increasingly sophisticated attacks, creating a gap between system complexity and analysis capability. My vision is to bridge this gap by *developing security analysis that evolves as rapidly as the systems it protects*. To achieve this, I will pursue three progressive directions: *reasoning* about the probabilistic semantics of AI-driven behaviors, *unlocking* compositional properties for system-wide guarantees, and *closing the loop* between vulnerability discovery and automated defense.

Probabilistic Reasoning for Non-Deterministic Systems. Software is evolving beyond traditional deterministic paradigms to incorporate AI-driven decision-making, autonomous agents, and probabilistic reasoning. These systems exhibit emergent behaviors that traditional security analysis cannot adequately capture. For example, agentic systems where LLM-driven agents interact with traditional code create subtle cross-component risks invisible to conventional analysis. Leveraging my experience in coordinating LLMs with program analysis [8, 9], I will develop *probabilistic abstract interpretation* to reason about these non-deterministic behaviors mathematically. Instead of tracking concrete values, my abstractions will track probability distributions over program states, treating LLM interactions as stochastic processes to map the infinite state space into manageable, abstract domains. This allows us to mathematically bound non-deterministic behaviors and verify safety properties even in worst-case scenarios.

Compositional Verification for Heterogeneous Scale. While probabilistic reasoning addresses *what* we analyze, we must also transform *how* analysis scales and adapts. Future security analysis must achieve fundamental properties that current techniques lack, *e.g.*, accessibility for broad adoption, resilience to software evolution, and provability for formal guarantees. My future research will apply my philosophy to unlock these important properties. For example, by enabling independent reasoning about components, I aim to achieve accessibility where developers can verify local services without mastering the entire global system complexity. This independence also fosters resilience through incremental verification, allowing system updates to trigger re-analysis of only specific component summaries rather than the entire codebase. Furthermore, I will ensure provability by automatically composing these local properties to derive rigorous system-wide security guarantees. I will develop compositional abstractions that capture these local properties and incremental algorithms that preserve them, transforming security analysis from a brittle process into a resilient foundation.

Achieving Autonomous Security Evolution. The ultimate vision is security systems that evolve autonomously, discovering emerging vulnerability classes, synthesizing defenses, and continuously improving without human intervention. Current security practices still require excessive manual engineering for each of these steps. My research will establish a new paradigm where security systems evolve across the *entire security lifecycle*. I will develop foundations that enable autonomous evolution through neuro-symbolic reasoning, where generative models propose defenses and formal verifiers enforce safety constraints before deployment. Specifically, I will frame defense generation as a constrained synthesis problem [11], leveraging the generative capabilities of LLMs to explore novel mitigation strategies while using symbolic execution to strictly prune unsafe candidates. For example, systems might autonomously learn vulnerability principles from attack patterns and synthesize patched binaries that are mathematically proven to preserve system functionality. Crucially, this creates a continuous feedback loop where successful defenses are distilled into generalized policies, allowing the analysis engine to refine its own reasoning logic and adapt to future threats. This transforms security from a reactive, manual effort into a self-improving, autonomous ecosystem that scales with the threat landscape.

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.