# Research Statement

Penghui Li

My research interests lie in *software security, program analysis, and software engineering*. Software underpins critical infrastructure in everyday life, yet modern software systems have become increasingly complex in scale and heterogeneity. Sophisticated adversaries routinely exploit vulnerabilities in production software, leading to system compromises, infrastructure disruptions, and economic damages. Despite decades of progress, current security practices rely on *engineered automation* where security experts encode their domain knowledge into automated security analysis tools that follow fixed, pre-defined heuristics. This approach is often costly to build, brittle to software changes, and rarely generalizes across codebases. My research vision is to *shift security analysis from engineered automation to autonomy*, where analysis can independently reason about software security across heterogeneous systems at scale and continuously adapt its defenses to the evolving threat landscape.

My research philosophy is to drive security analysis through *principled abstraction and decomposition*. Abstraction eliminates irrelevant complexity to reveal essential security properties, while decomposition breaks intractable challenges into solvable sub-problems. By choosing an appropriate abstraction layer and decomposition granularity, my research identifies critical bottlenecks in security practices and develops deployable defenses with accessibility and impact. For instance, to detect privilege escalation in polyglot microservices, I abstracted heterogeneous service implementations into a unified model of cross-service interactions, and decomposed the analysis task into complementary components handled by rigorous program analysis and semantic-aware large language model (LLM) reasoning. This realizes a novel *agentic program analysis* approach, where autonomous agents orchestrate specialized techniques to discover and validate complex threats. Looking forward, I will apply this philosophy to secure emerging software paradigms, discover novel threat classes, and enable autonomous defense.

My research has made significant real-world impacts. I have applied my philosophy to securing web applications [1, 2, 3, 8], cloud software [4], desktop applications [5, 9], operating systems [10], and database systems [11]. 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 promptly to protect millions of users worldwide. My research has been published at top-tier venues in security (S&P, Security, CCS, NDSS) and software engineering (ICSE, FSE, ASE), and has been recognized with a Distinguished Paper Award at CCS 2024, a Best Paper Honorable Mention at CCS 2022, and a Distinguished Artifact Award at CCS 2025.

## Past and Current Work

Software systems are ultimately built to serve users' needs and directly influence their security, privacy, and access to critical services. I apply my research philosophy to target domains where vulnerabilities have the most direct and widespread impact on users' daily lives. My research starts from monolithic web applications that serve as the primary interface for online services, and extends to distributed cloud-native microservice systems. Building on these concrete challenges in scale and heterogeneity, I advance foundational program analysis techniques that improve the scalability and precision of security analysis in general.

**Language-Based Security for Web Applications.** Web applications are predominantly developed in dynamic languages such as PHP, JavaScript, and Python, which prioritize rapid development and flexibility but introduce unique security challenges. The fundamental challenge to program analysis lies in understanding and supporting complex language features, including weak type systems and intricate language features. I develop language-based static code search and dynamic test generation techniques to systematically explore and validate vulnerabilities.

I view the task of identifying vulnerable code components in static analysis as a *code search problem* that retrieves program elements satisfying specific criteria. My core methodology is to separate the representations of the program and the search criteria. Specifically, I abstract language-specific complexities into a code database and develop a domain-specific language (DSL) framework where search tasks are expressed as simple, declarative queries.

Unlike prior approaches that require complex graph query languages [14], my DSL distills vulnerability detection into four elementary operation types (name lookup, operation lookup, call graph traversal, and data-flow tracking), each corresponding to a fundamental program analysis task. These operations, extracted from analyzing existing vulnerability patterns, cover over 90% of common detection tasks and can be systematically composed to express complex vulnerability patterns. This approach uncovered over *60 previously unknown vulnerabilities*, including 25 CVEs, and revealed *a design flaw in PHP's type system* that influenced PHP 8.0 [3, 8].

While static analysis reasons about code structure and potential behaviors, dynamic analysis exposes actual execution behaviors that are only observable at runtime, which is particularly critical for dynamic web languages. Many prior solutions focus on improving algorithmic strategies for path exploration. However, my systematic profiling of real-world web applications revealed that execution efficiency is actually the performance bottleneck. Specifically, accesses to external resources such as databases and network services dominate execution time (around 50%), as each test iteration re-fetches the same data and even small slowdowns compound across repeated executions. I developed a software-based data caching mechanism that stores frequently accessed data in shared memory and a just-in-time code compilation technique, reducing redundant operations and improving execution throughput by up to 4 times [2]. This solution is generic and applicable to any dynamic analysis tool as a plug-in component, and has helped discover new vulnerabilities in *WordPress*, which powers *nearly half of all websites worldwide*.

**Agentic Analysis of Microservice Systems.** Modern applications increasingly adopt microservice architectures in the cloud, where applications are decomposed into distributed services. These systems are often polyglot, with different services in different languages or frameworks. This introduces fundamentally new security challenges as vulnerabilities arise from subtle cross-service interactions, and traditional single-codebase analysis tools cannot trace end-to-end attack paths. Moreover, security requirements in microservices are often implicit, encoded in natural language documentation rather than explicit code, making it difficult to identify relevant security-critical code paths.

I developed an *agentic program analysis* approach that autonomously analyzes microservice systems by integrating LLM reasoning with my DSL-based program analysis framework [4, 6]. The two components work synergistically: LLMs interpret implicit security requirements from documentation and API specifications, formulate analysis strategies, and reason about cross-service authorization logic. Meanwhile, DSL-based program analysis provides systematic code search and precise data flow tracing across services, validates LLM-generated hypotheses against actual code, and discovers concrete attack paths. For instance, when an LLM infers from documentation that a service endpoint requires authorization, program analysis validates this by tracing call chains to verify whether authorization checks exist along execution paths. Conversely, when program analysis identifies a data flow crossing service boundaries, LLM reasoning interprets the security implications by analyzing related authorization policies. The autonomous agents orchestrate these capabilities, decomposing complex cross-service analysis into sequences of steps combining both LLM reasoning and rigorous program analysis.

I applied this approach to detect well-known vulnerabilities such as SQL injection and remote code execution, and advanced cross-service privilege escalation attacks that exploit authorization gaps across service boundaries. The approach automatically decomposes analysis per language and service, then chains results across boundaries to reconstruct complete attack paths. It has discovered 20 critical privilege escalation vulnerabilities in production cloud applications that existing program analysis tools and LLMs alone could not detect.

**Foundations of Program Analysis.** I advance foundational program analysis techniques with higher scalability and precision. In symbolic execution, a core challenge is its reliance on manually-written models of complex language features, which are often incomplete and incorrect. I first developed a differential testing framework that uncovered widespread correctness bugs in the models of mature symbolic execution engines [7]. To eliminate this entire class of errors, I then developed holistic symbolic interpreter analysis [1], a technique that bypasses manual models altogether. Since all language features are implemented within the interpreter itself, by symbolically executing the interpreter, my approach automatically supports the complete language syntax with guaranteed fidelity. This foundational advance ensures correctness by construction, reduces engineering effort from over a year to under two weeks, and enables the precise security analyses for dynamic interpreted languages and applications.

I also improve directed greybox fuzzing, a testing technique that targets specific program locations to efficiently

discover vulnerabilities. I developed new approaches [5, 12] that refine testing directions while reducing the exploration space. By extracting information from static analysis and guiding exploration through intelligent runtime state management, my approach significantly reduces wasted exploration on infeasible paths. This achieved 105% faster vulnerability exposure overall.

## Future Directions

Building on my prior research foundations, my future research advances autonomous security analysis through three interconnected thrusts. These thrusts form a unified path toward fully autonomous security by first understanding emerging behaviors, then discovering novel threats, and finally enabling autonomous defense. By applying principled abstraction and decomposition to expose security properties and break down complex challenges, these directions will realize the shift to an autonomous security lifecycle.

**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 apply abstraction and decomposition to develop practical analysis solutions. Guided by my philosophy, 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. 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 [6] 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 apply abstraction and decomposition to systematically replicate this capability in automated systems. My approach will 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 apply decomposition to break 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, Hong Yau Chong, Yinzhi Cao, and Junfeng Yang. "Detecting Privilege Escalation in Polyglot Microservices via Agentic Program Analysis". Under Review.

[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, 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`.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[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] 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.