1 Introduction

Firmware is a strong target for vulnerability-discovery research because it combines high impact with persistent analysis difficulty. Embedded firmware has repeatedly shown broad and long-lived weaknesses [3]. In practice, source code is often unavailable, updates are slow, and binaries are heterogeneous across architectures/toolchains. Dynamic analysis is often difficult because it depends on firmware rehosting and emulation, where setup and coverage constraints remain substantial [2, 1]. This creates a gap between what is theoretically analyzable and what can be audited reliably under realistic time and compute budgets.

We focus on firmware because it is both security-relevant and a realistic stress case for practical analysis workflows. This is also why firmware is a good LLM target: if dynamic analysis is bottlenecked by rehosting/emulation, then a static-first workflow can create strong value as a triage and hypothesis-generation layer; dynamic confirmation can be optional or targeted. The core problem is firmware vulnerability discovery on binary-only inputs with heterogeneous tool outputs and incomplete context.

LLMs add value here when used as control components, not stand-alone detectors. Compared with traditional static pipelines, an LLM-driven framework can provide: (i) **cross-tool orchestration** (routing between disassembly, decompilation, xref, and string analyses under one plan), (ii) **cross-function and cross-binary reasoning** (tracking evidence across call chains and binaries instead of isolated function-level checks), (iii) **semantic hypothesis refinement** from pseudocode/assembly context, and (iv) **evidence-grounded iterative analysis** that updates priorities as new facts appear. Recent work supports parts of this direction: stripped-binary semantic recovery (SymGen, VulBinLLM) [6, 5] and tool-mediated planning loops in security agents [7]. The main risk is unreliability (hallucination, over-exploration, unstable decisions), which motivates strict execution constraints and direct comparison with non-LLM baselines.

Classical firmware/binary systems provide core primitives (rehosting, symbolic/concolic exploration, fuzzing), but not planner-level resource allocation for long workflows [2, 8, 11]. Concretely, when analysts face N binaries, M candidate functions, and K analysis tools (decompile/xref/taint/strings), these systems do not directly optimize which action should be taken next, when to stop, or when to pivot under weak evidence. For firmware static analysis specifically, Operation Mango demonstrates strong taint-style vulnerability discovery by scaling static dataflow analysis [4]. Recent LLM-binary papers improve semantic recovery, yet leave open reliability and comparative effectiveness on firmware-scale workflows [9, 6, 5]. Our work is positioned as a reproducible agent-control design evaluated against both LLM-based and traditional analysis baselines.

Research Questions. RQ1.

> *How can we design and build an LLM-driven framework (BinAgent) that effectively leverages planning, scheduling, and pseudocode-level reasoning to improve firmware binary analysis efficiency and end-to-end vulnerability discovery outcomes?*

RQ2.

> *How should we define a benchmark and evaluation protocol that meaningfully characterizes BinAgent's capabilities, limitations, and comparative performance against recent LLM-based systems and traditional static firmware-analysis workflows?*

Our working hypothesis is that the main bottleneck is not the lack of standalone analysis primitives, but weak orchestration across existing ones. We therefore design **BinAgent** as an

LLM-driven control layer that performs planning, tool routing/scheduling, pseudocode reading, and evidence-grounded synthesis over existing firmware-analysis tools.

These research questions motivate two design principles embedded in the introduction above: first, LLM-orchestrated integration of existing methods (planning, dispatch/scheduling, and pseudocode-guided reasoning) as the mechanism for improvement; second, explicit comparison against traditional workflows to explain where and why LLM-driven orchestration adds measurable value.

Planned contributions.

1. **BinAgent framework design:** a planner-centric LLM control layer that coordinates existing static-analysis and reverse-engineering tools through plan–act–observe loops, tool scheduling, and pseudocode-guided reasoning.

2. **Integration methodology:** a concrete recipe for combining heterogeneous analysis outputs (disassembly, decompilation, xrefs, strings, and notes) into evidence-linked vulnerability hypotheses and decisions.

3. **Benchmark specification for BinAgent:** a capability-oriented benchmark and protocol covering single-binary and multi-binary firmware tasks, with clear metrics, artifact requirements, and baseline definitions [10].

References

[1] Ioannis Angelakopoulos, Gianluca Stringhini, and Manuel Egele. Pandawan: Quantifying progress in linux-based firmware rehosting. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5859–5876, 2024.

[2] David D. Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for linux-based embedded firmware. In *23rd Annual Network and Distributed System Security Symposium (NDSS)*, 2016.

[3] Andrei Costin, Jonas Zaddach, Aurelien Francillon, and Davide Balzarotti. A large-scale analysis of the security of embedded firmwares. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 95–110, 2014.

[4] Will Gibbs et al. Operation mango: Scalable discovery of taint-style vulnerabilities in binary firmware services. In *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[5] Noman Hussain, Haotian Chen, Chi Tran, Peibin Huang, Zicheng Li, Prabh Chugh, and Yu Tian. Vulbinllm: Llm-powered vulnerability detection for stripped binaries. *arXiv preprint arXiv:2505.22010*, 2025.

[6] Linxi Jiang, Xin Jin, and Zhiqiang Lin. Beyond classification: Inferring function names in stripped binaries via domain-adapted llms. In *Network and Distributed System Security Symposium (NDSS)*, 2025.

[7] Xiangmin Shen, Lingzhi Wang, Zhenyuan Li, Yan Chen, Wencheng Zhao, Dawei Sun, Jiashui Wang, and Wei Ruan. Pentestagent: Incorporating llm agents to automated penetration testing. In *Proceedings of the 20th ACM ASIA Conference on Computer and Communications Security (ASIACCS)*, 2025.

[8] Nick Stephens, Josh Grosen, Christopher Salls, Nick Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Driller: Augmenting fuzzing through selective symbolic execution. In *23rd Annual Network and Distributed System Security Symposium (NDSS)*, 2016.

[9] Hengrui Tan, Qirui Luo, Jiaqi Li, and Yifei Zhang. Llm4decompile: Decompiling binary code with large language models. *arXiv preprint arXiv:2403.05286*, 2024.

[10] Ruozhao Yang, Mingfei Cheng, Gelei Deng, Tianwei Zhang, Junjie Wang, and Xiaofei Xie. Pentesteval: Benchmarking llm-based penetration testing with modular and stage-level design. *arXiv preprint arXiv:2512.14233*, 2025.

[11] Insu Yun, Sangho Lee, Meng Xu, Yeongjin Jang, and Taesoo Kim. Qsym: A practical concolic execution engine tailored for hybrid fuzzing. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.