Firmware Intro and Related Work

1 Introduction

Firmware remains a high-impact and persistent attack surface: it is widely deployed, tightly coupled to device-critical functionality, and often updated less frequently than user-space software. Auditing firmware binaries is hard in practice because source code is typically unavailable, toolchains/architectures are heterogeneous, and the analysis space becomes prohibitively large under realistic time and compute budgets. This creates a recurring gap between *theoretically analyzable* and *operationally auditable*.

This work studies whether an LLM-driven security agent can narrow that gap via structured planning plus tool-grounded binary analysis. We focus on vulnerability detection in stripped or partially symbolized binaries, where the agent must infer structure, prioritize risky regions, and justify findings with explicit evidence. Instead of brute-force tool invocation, we enforce a plan–act–observe–replan loop with bounded iterations, evidence-aware prioritization (dangerous APIs, cross-references, data-flow hints), and scalable decomposition (coarse triage followed by selective deep dives). The goal is to improve both effectiveness (finding real vulnerabilities) and efficiency (latency and tool/token cost), while preserving auditability through machine-checkable evidence trails.

Our research question is: *How can a security agent expand the range of binary vulnerability tasks it can solve from minimal task descriptions, while remaining fast and reliable on large firmware inputs, using principled planning and evidence-guided decomposition?* To answer this, we develop **BinAgent**, built on top of **PentestAgent**, and evaluate it on progressively harder reverse-engineering and vulnerability-analysis tasks. The intended contribution is a practical control-layer architecture for firmware-scale binary auditing that is measurable, reproducible, and extensible across analysis backends.

Planned contributions.

1. A planner-centric agent architecture for firmware binary auditing that integrates LLM reasoning with static analysis/decompiler tooling, and requires evidence-linked outputs.

2. An evidence-aware prioritization policy for large binaries that ranks risky functions/callsites before deep inspection, reducing unnecessary exploration.

3. A scalable decomposition workflow (coarse-to-fine analysis, selective deep dives, caching/memory control) designed for latency and cost constraints.

4. An artifact-driven evaluation protocol (plans, tool logs, evidence traces, outcomes) to quantify speed–quality tradeoffs and failure modes.

2 Related Work

2.1 Firmware Binary Analysis

Foundational firmware work established scalable pipelines for extraction, rehosting, and dynamic analysis. Large-scale studies highlighted ecosystem-wide weaknesses [4]. FIRMADYNE introduced automated dynamic analysis for Linux-based firmware images [3], and later systems such as FirmAE improved practical emulation success rates [9]. More recent efforts broaden coverage across user/kernel boundaries and device behaviors, including FirmSolo (kernel modules), Pandawan (holistic rehosting progress), and Laelaps/FFXE-style advances in peripheral modeling and control-flow recovery [1, 2, 11, 18]. These works are essential for execution coverage, but they do not directly solve planner-level prioritization under strict latency budgets. Our work complements them by optimizing *what to analyze next* when full emulation context is absent or too expensive.

2.2 Binary Vulnerability Discovery and Program Analysis

Classical binary vulnerability discovery combines symbolic execution, concolic execution, and fuzzing. Driller demonstrated selective symbolic execution as a practical accelerator for fuzzing [16], and QSYM further emphasized performance-aware hybrid concolic design [22]. Cross-architecture bug discovery systems (e.g., discovRE) and SoK analyses formalize why selective expensive reasoning often outperforms exhaustive search [6, 15]. Our design inherits this principle at the agent-control layer: use cheap global scans to

rank hypotheses, then spend expensive tool calls only on evidence-supported regions.

2.3 LLMs for Binary Analysis

Recent work shows that LLMs can contribute directly to binary workflows. LLM4Decompile demonstrates strong LLM-assisted decompilation capability and highlights the importance of functional/executability-oriented evaluation [17]. VulBinLLM targets stripped-binary vulnerability detection with decompilation optimization and long-context memory strategies [10]. These results motivate our setting, but leave open the systems question of *reliable orchestration*: how to keep tool use bounded, evidence-linked, and robust as binary scale increases.

2.4 LLMs and Tool-Using Agents for Security Tasks

Tool-using LLM agents have shown strong improvements when reasoning is interleaved with external actions [21, 13]. Planning-centric variants (Plan-and-Solve, Tree-of-Thoughts), critique/reflection loops (Reflexion, CRITIC), and memory hierarchies (MemGPT) provide design patterns for long-horizon reliability [19, 20, 14, 8, 12]. In security contexts, PentestGPT and autonomous hacking studies show promise, but also surface reproducibility and safety/reliability limits on complex tasks [5, 7]. Our approach operationalizes these lessons with mandatory planning, bounded loops, and strict evidence linkage (e.g., CWE + function/address + snippet + tool provenance).

2.5 Positioning of This Work

This project sits at the intersection of firmware RE, binary analysis, and LLM agent orchestration. The novelty is not a new decompiler, emulator, or symbolic engine; it is a control-layer design that (i) adapts analysis depth to observed evidence, (ii) prioritizes dangerous regions before broad exploration, (iii) manages context/cost via scalable decomposition and memory control, and (iv) outputs machine-checkable artifacts suitable for regression testing. This framing makes BinAgent a practical vehicle for studying scalability and reliability in firmware-oriented binary auditing.

Citation placeholders. Update and verify bibliography entries before submission: `costin2014large`, `chen2016firmadyne`, `kim2020firmae`, `angelakopoulos2023firmsolo`, `angelakopoulos2024pandawan`, `mukherjee2020laelaps`, `tsang2024ffxe`,

stephens2016driller, yun2018qsym, eschweiler2016discovere, shoshitaishvili2016sok, tan2024llm4decompile, liu2025vulbinllm, yao2022react, schick2023toolformer, wang2023planandsolve, yao2023treeofthoughts, shinn2023reflexion, gou2023critic, packer2023memgpt, deng2023pentestgpt, fang2024autonomoushacking.

References

[1] Konstantinos Angelakopoulos, George Cretu, Alexandros Voyiatzis, and Vasileios Pappas. Firmsolo: Enabling dynamic analysis of binary linux-based iot kernel modules. In *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[2] Konstantinos Angelakopoulos, George Cretu, Alexandros Voyiatzis, and Vasileios Pappas. Pandawan: Quantifying progress in linux-based firmware rehosting. In *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[3] 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.

[4] Andrei Costin, Apostolis Zarras, and Aurelien Francillon. A large-scale analysis of the security of embedded firmwares. In *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[5] Ge Deng, Y Liu, Victor Mayoral-Vilches, et al. Pentestgpt: An llm-empowered automatic penetration testing tool. *arXiv preprint arXiv:2308.06782*, 2023.

[6] Sebastian Eschweiler, Khaled Yakdan, and Elmar Gerhards-Padilla. discovre: Efficient cross-architecture identification of bugs in binary code. In *23rd Annual Network and Distributed System Security Symposium (NDSS)*, 2016.

[7] Ruohong Fang, Pratham Bindu, et al. Llm agents can autonomously hack websites. *arXiv preprint arXiv:2402.06664*, 2024.

[8] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

[9] Donggeon Kim, Hyungjoon Kim, Minwook Choi, Daegyeong Kim, Yonghwi Kim, Suhwan Kim, Sang Kil Lee, and Taesoo Choi. Firmae: Towards large-scale emulation of iot firmware for dynamic analysis. In *Annual Computer Security Applications Conference (ACSAC)*, 2020.

[10] Y Liu et al. Vulbinllm: Llm-powered vulnerability detection for stripped binaries. *arXiv preprint arXiv:2505.22010*, 2025.

[11] S Mukherjee et al. Laelaps: Practical and scalable peripheral emulation in firmware rehosting. In *Annual Computer Security Applications Conference (ACSAC)*, 2020.

[12] Charles Packer, Vivian Fang, Shishir Patil, Kevin Lin, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[13] Timo Schick, Jane Dwivedi-Yu, et al. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

[14] Noah Shinn, Federico Labash, Ashwin Gopinath, et al. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

[15] Yan Shoshitaishvili, Ruoyu Wang, Christopher Hauser, Christopher Kruegel, and Giovanni Vigna. Sok: (state of) the art of war: Offensive techniques in binary analysis. *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

[16] 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.

[17] Hengrui Tan, Xi Wang, Shuai Wang, Yifei Zhang, et al. Llm4decompile: Decompiling binary code with large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

[18] Alan Tsang et al. Ffxe: Dynamic control flow graph recovery for embedded firmware binaries. In *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[19] Lei Wang, Weizhe Xu, Jing Liu, Yi Sun, et al. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.

[20] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

[21] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[22] 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.