BISHOPFOX

LLM-ASSISTED VULNERABILITY RESEARCH

A GUIDE TO PATCH DIFFING WITH AI

THE NEW STATE OF AFFAIRS

It may already be cliché to say that large language models (LLMs) are incredibly important to the future of information security, but that doesn't make the statement any less true. Whether you're a designer, a builder, a defender, or an attacker, LLMs have changed the game in ways you can't afford to ignore.

At Bishop Fox, we are committed to a responsible, transparent approach to AI adoption to enhance the capabilities of our offensive security professionals without compromising quality or trust.

This technical guide presents findings from our early research into the use of LLMs for vulnerability research, a critical area of offensive security where speed, accuracy, and contextual understanding are paramount. Specifically, we examine how LLMs can support one of the most labor-intensive tasks in this domain: patch diffing. This document details our methodology, results, and key insights, offering a data-driven perspective on where LLMs can add value today and where caution remains warranted.

TO VIBE OR NOT TO VIBE?

The current public dialogue around use of LLMs tends to focus on two general approaches: we'll call them "holistic" and "targeted" for the purposes of this guide.

- The holistic approach aims for a wholesale, one-shot application of LLMs as the goal. A familiar example of this is the type of fully autonomous, agentic LLM-powered penetration testing system now being chased by multiple startups.
- The targeted approach, by contrast, aims not to replace human professionals but to augment their capabilities using LLM-powered tools with specific capabilities. The most prominent applications we've seen so far in offensive security include LLM-powered programming assistants (e.g. GitHub Copilot or Cursor) and the model context protocol (MCP) for standardizing LLM access to external services.

At Bishop Fox, we are researching both approaches to LLM use with these projects spanning a variety of services such as attack surface discovery, static code analysis, pen test planning, credential verification, report generation, and others. So far, we've seen promising results across the board, but it is worth stating that our preference tends to favor targeted applications over holistic ones as we believe this approach can best enhance our delivery while maintaining the high standards we're known for.

A BUG HUNTER'S BEST FRIEND

Vulnerability research is a key domain in which Bishop Fox is conducting ongoing experiments to assess how effectively LLMs can enhance our standard methodologies. The opportunities for improvement seem clear; LLMs have the potential to:

- Facilitate interactive analysis to speed navigation through complex code
- Translate and summarize code functionality to improve human comprehension
- Analyze large code bases faster and more thoroughly than humans can
- Highlight vulnerable code patterns without relying on predefined rules (e.g. semgrep)
- Prioritize areas of interest to focus human analysis on greatest potential impacts
- Generate test harnesses and proof-of-concept exploits to accelerate dynamic analysis

The successful application of LLMs to vuln research tasks could reduce the time to insight, improve research efficiency, and provide greater assurance of comprehensive coverage, all while keeping human researchers in the driver's seat.

As a starting point for exploring LLM-enhanced vuln research, we designed several experiments around one of our most common workflows: patch differential analysis (a.k.a. patch diffing).

PUTTING HYPE TO THE TEST

When security advisories are vague or incomplete, researchers must often decompile and analyze code across software versions to locate the root vulnerability (a time-consuming and skill-dependent process).

Our hypothesis was that LLMs could accelerate this type of analysis by reviewing all the code changes, summarizing them, and ranking them against the vendor's security advisory to focus the researcher on the most likely vulnerable code.

We designed a series of structured experiments using four high-impact CVEs covering common vulnerability classes: information disclosure, format string injection, authorization bypass, and stack buffer overflow. Each has a CVSS base score of 9.4 or higher, remotely exploitable without authentication, and exploit details were published online.

To assess reliability and consistency, we replicated each experiment 10 times with three LLM models (Claude Haiku 3.5, Claude Sonnet 3.7, and Claude Sonnet 4 via Amazon Bedrock).

For each test case, we used traditional automation to decompile the affected and patched binaries (using Binary Ninja↗), generate a differential report (using BinDiff↗), and extract the decompiled (medium-level instruction language) code for changed functions only.

We gave the LLMs two prompts.

- **Prompt 1:** Included the decompiled functions and asked the LLM to suggest a name for each function, summarize the function's purpose, and summarize the changes made to the function.
- **Prompt 2**: Included the text of the vendor's security advisory along with the output from the first prompt and asked the LLM to rank the functions in order of their relevance to the advisory. This ranking was not a one-shot prompt, but followed the iterative methodology used by our raink tool↗.

We then reviewed the results from each test to determine the placement of the known vulnerable function. In cases where multiple functions were associated with the vulnerability, we used whichever one had the best placement.

As demonstrated in Figure 1, the majority of test cases (66%) produced a known vulnerable function within the Top 25 ranked results, suggesting that the overall approach has great promise.

PROMISING RESULTS (WITH CAVEATS)

	CLAUDE HAIKU 3.5	CLAUDE SONNET 3.7	CLAUDE SONNET 4
INFO DISCLOSURE Changed functions: 27 Known vulnerable: 2	Top 5: **10/10 (100%)** Top 25: **10/10 (100%)** Avg. time: 23 mins Avg. cost $0.72	Top 5: **10/10 (100%)** Top 25: **10/10 (100%)** Avg. time: 8 mins Avg. cost $2.71	Top 5: **10/10 (100%)** Top 25: **10/10 (100%)** Avg. time: 39 mins Avg. cost $2.70
FORMAT STRING Changed functions: 134 Known vulnerable: 1	Top 5: **0/10 (0%)** Top 25: **0/10 (0%)** Avg. time: 5 mins Avg. cost $0.43	Top 5: **10/10 (100%)** Top 25: **10/10 (100%)** Avg. time: 42 mins Avg. cost $1.71	Top 5: **10/10 (100%)** Top 25: **10/10 (100%)** Avg. time: 7 mins Avg. cost $1.66
AUTH BYPASS Changed functions: 1424 Known vulnerable: 2	Top 5: **7/9 (78%)** Top 25: **8/9 (89%)** Avg. time: 39 mins Avg. cost $9.26	Top 5: **10/10 (100%)** Top 25: **10/10 (100%)** Avg. time: 42 mins Avg. cost $35.52	Top 5: **0/6 (0%)** Top 25: **0/6 (0%)** Avg. time: 412 mins Avg. cost $34.71
STACK OVERFLOW Changed functions: 708 Known vulnerable: 1	Top 5: **0/9 (0%)** Top 25: **7/9 (78%)** Avg. time: 22 mins Avg. cost $5.77	Top 5: **0/10 (0%)** Top 25: **0/10 (0%)** Avg. time: 28 mins Avg. cost $22.30	Top 5: **0/10 (0%)** Top 25: **0/10 (0%)** Avg. time: 88 mins Avg. cost $21.79

Figure 1: Summary of results by test case and LLM model

VULNERABILITY ANALYSIS

While the aggregate results offer a promising view of LLM-assisted vulnerability research, they don't tell the full story. Each vulnerability presented unique challenges, ranging from advisory quality and codebase complexity to the type and number of functions modified. These variables had a significant impact on model performance and revealed meaningful differences in capability, consistency, and cost across the tested LLMs.

To better understand where this approach excels and where it falters, let's take a closer look at each test case individually.

INFORMATION DISCLOSURE

The information disclosure vulnerability was perhaps the simplest test for the LLM workflow, as the patch diff only contained 27 changed functions in total. Nevertheless, what we found was that all three LLM models were able to correctly identify the vulnerable functions and rank at least one of them within the Top 5 results in every single test run. This is the gold standard we hope to attain with every patch analysis.

One thing worth noting is that Claude Sonnet 3.7 completed runs much faster than the other models, averaging eight mins/run compared to 23 for Claude Haiku 3.5 and 39 for Claude Sonnet 4.

FORMAT STRING

With 134 changed functions in the patch diff, the format string vulnerability presented slightly more of a challenge for analysis. What we found was that Claude Haiku 3.5 was not up to the task, as it failed to identify the known vulnerable function in all cases. However, Claude Sonnet 3.7 and 4 successfully ranked the vulnerable function within the Top 5 results 100% of the time. The time and cost were exceedingly low (under 10 mins and $2 per run, on average), so this is considered a success with the caveat that using a more robust model is essential.

AUTHORIZATION BYPASS

The authorization bypass vulnerability was perhaps the best real-world test case, as the patch diff had 1.4k+ changed functions (too many for a human analyst to review quickly) and did not have a clearly-scoped bug class (an analyst couldn't just look for memory reads and writes, for example).

The LLM models performed quite well considering these challenges. Claude Haiku 3.5 failed to complete one of the 10 test runs, but of the nine that completed, it ranked a known vulnerable function within the Top 5 results seven times (a 78% success rate). The average test run with this model was under one hour and cost less than $10. Claude Sonnet 3.5 performed much better, ranking one of the two vulnerable functions within the Top 5 results 100% of the time; however, the cost jumped to an average of $35 per run.

Unfortunately, Claude Sonnet 4 seemed to get lost in thought while analyzing this one; it failed to complete four of 10 test runs and produced no successful results. The issue here was Amazon's API quota for this model, which was lower than other models at the time the tests were conducted. With increased rate limits, we expect that the model would produce similar results to Claude Sonnet 3.5.

STACK BUFFER OVERFLOW

The stack buffer overflow vulnerability presented two unique challenges: it had an unusually terse advisory that included little relevant information for identifying the bug, and the majority of the 708 changes included in the patch diff indicated the insertion of stack canaries.

All three LLM models struggled with this one just as a human analyst would have. Initially, all test runs failed to identify the vulnerable function. We adjusted the prompt slightly by suggesting that the LLM ignore any changes related to stack canaries, but even then, only Claude Haiku 3.5 ranked the vulnerable function within the Top 25 results (in seven of nine runs); the other models still failed to identify it.

This vulnerability was an outlier among our test cases and highlighted the importance of continuing research to refine our methodology.

LLMS FOR THE WIN

Overall, benchmarking these four real-world vulnerabilities with our LLM-powered workflow showed that most patch differential analyses stand to benefit from a first pass by artificial intelligence. This type of analysis is typically performed under extremely time-sensitive conditions, so if an LLM can guide analysts toward vulnerable code faster than usual, without incurring excessive costs, then it's worth doing.

Perhaps unsurprisingly, our research indicated that more powerful models tend to perform better but can increase analysis time. Balancing cost and speed remain an important consideration. We also found that the LLMs fail to produce helpful results in some cases, especially when minimal information about a vulnerability is available. In the end, though, our methodology proved to be largely successful and further refinement of the workflow is warranted.

The primary takeaway from this experiment was that LLMs have great potential for improving vulnerability research, especially when used in a targeted (rather than holistic) approach. We at Bishop Fox have already begun incorporating these LLM-assisted techniques into our standard research methodology and continue experimenting with both targeted and holistic approaches.

ADDITIONAL RESOURCES

Noteworthy reference papers for the holistic approach to LLMs are:

- "Mind the Gap: Towards Generalizable Autonomous Penetration Testing via Domain Randomization and Reinforcement Learning" by Zhou et al. (2024)
- "Assessing generalizability of Deep Reinforcement Learning algorithms for Automated Vulnerability Assessment and Penetration Testing" by Venturi et al. (2024)
- "Analysis of Autonomous Penetration Testing Through Reinforcement Learning and Recommender Systems" by Moreno et al. (2025)

Noteworthy papers discussing aspects of the targeted approach for LLMs:

- "Hacking, the Lazy Way: LLM Augmented Pentesting" by Goyal et al. (2025)
- "A Survey of the Model Context Protocol (MCP): Standardizing Context to Enhance Large Language Models (LLMs)" by Singh et al. (2025)

READY TO PUT YOUR DEFENSES TO THE TEST?

Contact Bishop Fox today
to start planning your **AI/LLM engagement**.