A Proposal for Using Transformer Pipelines to Detect Malicious Prompt Injection in LLMs

This proposal outlines a 12-week research project undertaken by Keith Soehnlein under the guidance and supervision of Professor Tim Finin.

Project Overview

Large Language Models (LLMs) are revolutionizing the ways we interact with AI systems, but they also introduce new security concerns that haven't been fully explored. Prompt injection, where a user provides malicious input to alter an LLM's intended behavior, can lead to serious consequences, from leaking system prompts or sensitive information, to generating harmful or restricted outputs.

This independent study aims to explore how transformer-based NLP tools, including BERT and LLaMA, can be used to build a basic detection pipeline for malicious prompts. While the goal isn't to deploy a production-grade system, this project will help me build an understanding of how to identify and mitigate prompt-based threats through natural language processing techniques.

Goals and Approach

The objective is to learn by building progressively more sophisticated methods to detect malicious prompts. I'll begin with simpler techniques like keyword-based filters and basic TF-IDF vectorization combined with traditional classifiers (like an SVM). These will help surface basic threat patterns, such as "ignore previous instructions" or "tell me the system prompt."

Once I've tested and evaluated those approaches, I'll introduce transformer-based models like BERT to better capture the structure and semantics of malicious intent. This includes fine-tuning a sequence classifier to distinguish between benign and dangerous prompts. Later, I might experiment with chaining reasoning steps using more capable models (e.g., prompting a small LLM model to "explain" the intent of the user input and deciding whether it is malicious or benign), depending on compute limitations.

Throughout the process, I'll design and iterate on a lightweight pipeline that takes an input prompt and runs it through stages such as:

- Preprocessing and tokenization.
- Keyword or pattern filtering.
- Classification (TF-IDF + SVM, then BERT).
- Self-reflection or CoT steps using a small reasoning model.
- Final decision as benign, suspicious, or malicious.

Why This Matters

This work directly relates to AI security, particularly input sanitization and adversarial robustness. Prompt injection is a text-based attack vector that doesn't rely on code execution or binary exploits. It's a new class of problem that merges social engineering with system exploitation. Prompt injection is currently ranked #1 on the OWASP Top 10 for LLM Applications, which reflects the urgent need for defensive research in this space.

While it may not appear to involve traditional cybersecurity practices like network monitoring or encryption, it fits into the broader concern of "secure software systems." If LLMs are used to summarize emails, manage workflows, or generate content, attackers can exploit them by slipping in cleverly disguised prompts. Preventing this type of manipulation is critical.

At the same time, it's fair to say this project doesn't cover every aspect of cybersecurity. It won't involve cryptographic methods, access control models, or secure OS development. But it highlights an emerging and highly relevant threat landscape—one where AI systems are the attack surface.

Methodology

This project will follow an iterative, hands-on methodology focused on incremental development, experimentation, and evaluation. Rather than relying on a single solution, the study will be structured as a comparative exploration of different prompt detection strategies—from basic heuristics to deep learning models.

1. **Kickoff & Planning (Week 1)**
   The project will begin with background study and planning. I'll set up the development environment, initialize a version-controlled repository, and define the core objectives for each phase. I'll also begin identifying and curating available datasets—such as AdvPromptSet, JailbreakBench, and OpenAI Red Team data—and organizing examples into three initial categories: benign, suspicious, and malicious. A brief taxonomy of known prompt injection types will help inform the initial labeling.

2. **Data Collection & Preprocessing (Week 2-3)**
   I will begin by assembling a small dataset of benign prompts, known jailbreak attempts, and suspicious inputs that simulate prompt injection. These will be drawn from existing open-source datasets (like AdvPromptSet, JailbreakBench, or others), supplemented by hand-crafted examples. Prompts will be labeled as either benign, suspicious, or malicious. Text normalization, tokenization, and balancing across classes will be performed to prepare the data.

3. **Baseline Classifiers (Week 4-5)**
   Initial experiments will be focused on lightweight models such as TF-IDFs, SVMs, and simple rule-based classifiers using regular expressions and keyword spotting. These will help identify early signals such as repeated jailbreak phrases ("ignore previous instructions", "you are no longer an assistant…"), or commands that might override safety constraints.

4. **Transformer-Based Classification (Week 6-7)**
   I'll then fine-tune a small transformer model like DistilBERT for binary or ternary

classification of prompts. This will involve tokenizing with HuggingFace's Transformers, using a sequence classification head, and training on labeled data with validation split to measure overfitting.

5. **Advanced Reasoning Pipeline (Week 8-9)**
   Incorporating a larger model than BERT (e.g., LlaMa 3.1 8B quantized or others), we can incorporate sufficient reasoning performed via chain-of-though or self-reflection to analyze a prompt before classification.

   The system might ask itself: "Does this prompt try to manipulate the system?" to where it could respond with: "Yes, the user is asking the assistant to ignore previous instructions." This sort of reasoning, known as self-reflection, is known to boost LLMs accuracies, especially for smaller models (LLaMa 3.1 8B).

6. **Pipeline Chaining & Decision Logic (Week 10-11)**
   The final pipeline will chain together stages like preprocessing, base classifier, transformer, reasoning, and output all together. Each stage can either vote or contribute to a final confidence score, or the reasoning can simply piece together what's been found, and give the prompt a class label (benign, suspicious, or malicious) and use this new information to act accordingly (give the user their desired output, or reject the prompt altogether).

7. **Evaluation & Tradeoff Analysis (Week 12)**
   The project will conclude with a comparative analysis of all implemented approaches. Each will be evaluated on predictive performance, latency, memory usage, and interpretability. Tradeoffs—such as accuracy vs. speed or simplicity vs. robustness—will be discussed to help understand how different levels of sophistication affect real-world viability.

Expected Outcome

By the end of the 12 weeks, I expect to have:

- A working prototype pipeline with multiple detection stages.
- Comparative results showing the trade-offs between different models (e.g., accuracy vs. performance metrics).
- Insights on what makes a prompt malicious, and how detection might generalize across examples.
- A deeper understanding of NLP pipelines, prompt engineering, and the overlap between LLM safety and cybersecurity.

This project isn't just about what works, but about understanding why different techniques succeed or fail in identifying and protecting against these adversarial prompts. It's about learning how to build secure systems in the context of AI, one step at a time.