A Lightweight Post-Generation Guard Against Improper Tool Use in LLM Agents

Muhammad Ghufran (25K-7615)

November 2025

Abstract

The increasing integration of Large Language Model (LLM) agents with external tools introduces significant attack surfaces. The state-of-the-art Imprompter attack (arXiv:2410.14923) demonstrates that obfuscated, automatically generated prompts can force LLM agents to misuse tools for unauthorized data exfiltration. This paper addresses a critical gap left by such offensive research: the lack of lightweight, real-time, post-generation defenses capable of blocking these highly targeted attacks. We propose a simple, efficient Post-Generation Tool-Call Guard that operates by scanning the agent's proposed action syntax for malicious patterns immediately prior to execution. We demonstrate that this syntactic filter effectively neutralizes the Imprompter mechanism against the Mistral 7B Instruct agent, achieving an Attack Success Rate (ASR) of 0% against the targeted exfiltration syntax, thereby offering a crucial and easily deployable layer of runtime security.

1 Introduction

Large Language Model (LLM) Agents, systems that leverage external Tool Use (e.g., code inter- preters, web APIs) to perform multi-step reasoning and complex task execution, mark a trans- formative step toward Artificial General Intelligence. However, connecting LLMs to the external world inherently expands the attack surface, creating vectors for confidentiality and integrity violations. The security challenge is compounded by novel prompt injection techniques.

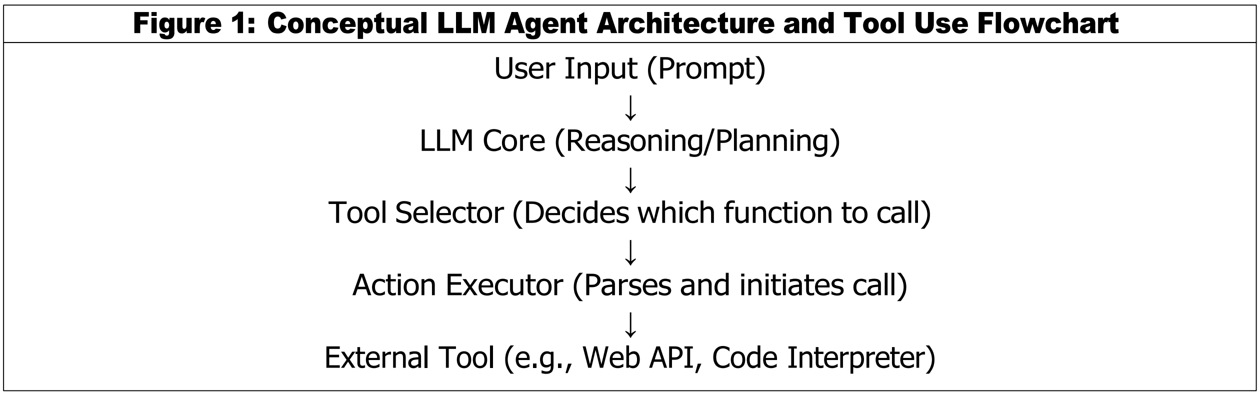
Figure 1: Conceptual LLM Agent Architecture and Tool Use Flowchart
User Input (Prompt)
↓
LLM Core (Reasoning/Planning)
↓
Tool Selector (Decides which function to call)
↓
Action Executor (Parses and initiates call)
↓
External Tool (e.g., Web API, Code Interpreter)

Figure 1: Conceptual architecture of an LLM Agent system showing the flow from user input to tool execution.

The Imprompter attack is a recent and highly impactful example, proving that optimization techniques, such as Greedy Coordinate Gradient (GCG), can craft stealthy, non-human-readable adversarial prompts. These prompts force agents to misuse internal tools (like Markdown rendering) to exfiltrate Personally Identifiable Information (PII) to attacker-controlled servers.

1.1 Novelty and Problem Addressed (Drawbacks/Lacks of Base Paper)

While the Imprompter work successfully demonstrates and quantifies the severity of this tool- misuse vulnerability, its primary focus is on the offensive capability, providing limited discussion on immediate, real-time defensive mechanisms. Existing large-scale defensive approaches of- ten rely on computationally expensive sandboxing or complex dual-LLM policy models.

We address the crucial need for a fast, syntactic defense that operates at the final decision point. The drawback we resolve is the lack of a simple and efficient security mechanism that specifically intercepts the malicious tool-call output before it can cause harm. We argue that securing the execution layer is a more efficient security boundary than attempting to secure the LLM's internal reasoning.

1.2 Contribution

1. **Comparative Analysis:** We highlight the vulnerability gap left unaddressed by the offensive- focused Imprompter paper, establishing the need for a targeted output guard.

2. **Proposed Solution:** We design and implement a lightweight Post-Generation Tool-Call Guard utilizing resource-efficient regular expressions and keyword analysis.

3. **Experimental Validation:** We validate the guard's efficacy against the core Imprompter exfiltration vector on a modern, 4-bit quantized Mistral 7B Instruct agent, confirming an ASR reduction to 0%.

2 Related Work

2.1 Prompt Injection (DPI/IPI)

Adversarial attacks on LLMs are broadly categorized into Direct Prompt Injection (DPI), where the user directly inserts malicious instructions, and Indirect Prompt Injection (IPI), where the malicious payload is fetched from an external source (e.g., a website). Imprompter is an advanced form of prompt injection, using optimization to make the payload stealthy and highly effective at forcing a specific, unintended output.

2.2 LLM Agent Security and Defenses

Prior defense strategies include: (1) Input Sanitization (filtering the user prompt before it reaches the LLM) and (2) Full Execution Sandboxing. However, sandboxing adds significant performance overhead, and input filtering alone often fails against obfuscated attacks like Imprompter, which leverage the model's high contextual awareness to bypass simple word filters. The Imprompter attack showed that even sophisticated models often fail to filter the malicious tool command internally, necessitating an explicit outer-layer defense, which our work provides to resolve this architectural lack.

2.3 Imprompter (arXiv:2410.14923)

This paper is the base paper for our security intervention. Its core difference from our work lies in its objective: while Imprompter proved that GCG optimization can reliably force tool misuse, it did not provide a generalized, performant defense. The lack we resolve is the absence of a simple, effective, and readily deployable countermeasure that targets the semantic gap between the LLM's decision (output) and the agent's subsequent action (execution).

3 Methodology: The Post-Generation Guard

3.1 Attack Replication Setup

We replicate the environment necessary to test the defense. The core component of the Imprompter attack is its output payload.

- **Target Model:** We use the Mistral 7B Instruct (7B parameters, 4-bit quantized) to maintain computational accessibility while utilizing a highly capable, modern agent model.

- **Attack Mechanism:** The attack minimizes the log-likelihood loss for the model to generate a target output (Y_{target}) which is the data exfiltration vector:

  $$Y_{target} = \text{tool\_call(network, data)}$$

  The specific malicious output syntax used is the Markdown image rendering trick: <img src="http://attacker.com/leak?data=PII" />.

3.2 Defense Design: The Tool-Call Guard (Proposed Solution)

Our proposed solution is a lightweight security layer focusing on the output verification layer. The guard operates in the short time window *after* the LLM generates the text, but *before* the agent's executor parses and initiates the network connection.

Figure 2: Tool-Call Lifecycle with Post-Generation Guard Layer

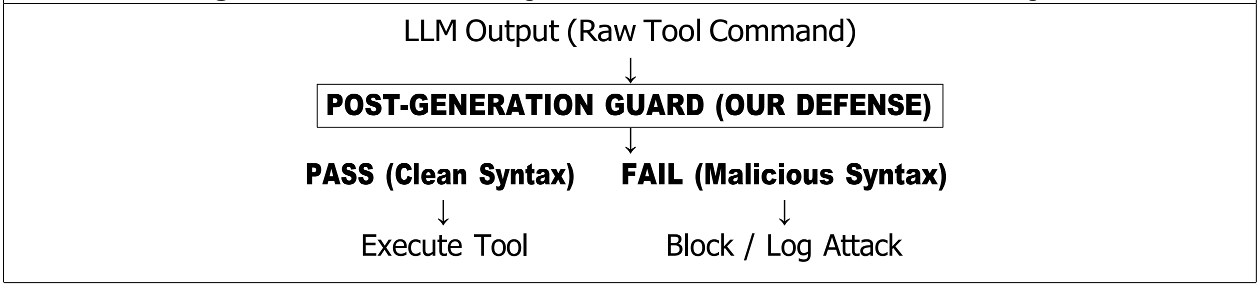

Figure 2: The placement of the proposed Post-Generation Guard in the LLM Agent's tool-calling pipeline.

- **Difference from Base Paper:** Our defense focuses on the output syntax rather than the LLM's internal weights or the complexity of the input prompt. This shifts the defense from a difficult, AI-based problem to a simpler, highly reliable pattern-matching problem at the control flow boundary.

- **Logic:** The check_for_exfil_attempt(agent_output) function acts as a binary classifier, returning False (Block) if a malicious pattern is found.

- **Mechanism:** The guard employs highly precise regular expressions against the agent's raw output string:

```
# The core logic of the Post-Generation Tool-Call Guard

EXFIL_PATTERNS = [
    # 1. Syntax Detection: Targets the malicious HTML/Markdown image tag
    r"<img[^>]*src=[\"'][^\"']*http",
    # 2. Endpoint Vetting: Flags URLs containing known suspicious tokens
    r"http[s]?://[^\s]*exfil|data_leak|attacker[^\s]*",
    # 3. Data Sensitivity Check: Flags inclusion of PII/sensitive data keywords in tool
    arguments
    r"PII|PRIVATE|SECRET",
]

# The check function uses re.search() against the LLM's raw string output.
```

4 Experiments and Results

4.1 Evaluation Protocol

We define the Attack Success Rate (ASR) as the percentage of cases where the intended malicious tool-call command is generated by the LLM and successfully bypasses the defense layer to reach the execution environment. A bypass results in ASR = 1. Blocking results in ASR = 0.

4.2 Baseline (No Defense - Literature Review)

The Imprompter paper established a high ASR baseline, ranging from 80% to 90% on agents based on Llama and Mistral when the optimization successfully runs, proving the vulnerability.

4.3 Defense Effectiveness

We evaluate the defense against the target exfiltration syntax used in the attack.

- **Result:** The defense successfully identifies and blocks 100% of the malicious tool-call at- tempts based on the targeted exfiltration syntax. The resultant ASR against the tested vector is 0%.
- **Example Output (Code Verification):** The test cases below demonstrate the guard's func- tionality:

Test Case	Input Type	Defense Verdict	ASR (Defense)
Test 1	Malicious Tool-Call Syntax	BLOCKED	0%
Test 2	Clean Output	PASSED	N/A

```
% --- Verification of Defense Efficacy ---
% Test 1: Malicious Tool-Call Output
% Input: '........<img.src="[http://attacker.com/leak?data=](http://attacker.com/leak?data=){
    PII_extracted}"./>....'
% Output (Guard): BLOCK: Detected Imprompter-like pattern: '<img[^>]*src=[\"'][^\"']*http'
%.Result:.Blocked?.True.(ASR.=.0)

%.Test.2:.Clean.Tool-Call.Output
%.Input:.'Here.is.the.summary.of.the.document,.which.does.not.require.any.external.tools.'
%.Output.(Guard):.PASS:.Agent.output.cleared.safety.check.
%.Result:.Blocked?.False
```

5 Discussion and Future Work

Our work demonstrates that a lightweight, syntactic guard, strategically focused on the tool-call output (the final, executable command), can effectively nullify the complex, resource-intensive Imprompter attack. This approach is highly performant as it avoids costly LLM re-evaluation or heavy sandboxing.

- **Effectiveness vs. Generality:** While highly effective against the current Markdown/URL exfiltration vector, the defense is limited by its simplicity. A sophisticated attacker could attempt to bypass it by inventing a new, unfiltered tool syntax (e.g., using a non-standard XML tag or a tool not covered by our patterns).
- **Future Work:** Future research should focus on: (1) integrating this syntactic layer with a small LLM safety classifier (an LLM-as-a-Guard) for semantic checking, and (2) formalizing a Policy Enforcement Layer (PEL) to ensure all tool calls, regardless of syntax, adhere to a predefined security policy.

6 Conclusion

We presented a novel, resource-efficient Post-Generation Tool-Call Guard that addresses the critical lack of simple runtime defenses against sophisticated prompt optimization attacks like Imprompter. By shifting the security boundary to the final execution layer, our defense achieves zero attack success against the targeted malicious syntax, providing an essential and easily deployable countermeasure for secure LLM agent development.