Available Online: 16 November 2026 DOI: 10.54254/2977-3903/2026.39089

Provenance-gated tool invocation for defending LLM agents against indirect prompt injection

Lulu Jing[1], *Nanmei Chen*[2], *Zhongmiao Sun*[3], *Chao He*[4], *Fan Si*[5,*]

[1]School of Management, Beijing Union University, Beijing, China
[2]Graduate School of Global Studies, Yonsei University, Seoul, Republic of Korea
[3]Shanghai Maritime University, Shanghai, China
[4]Graduate School of Technology Management, Kyung Hee University, Yongin, Republic of Korea
[5]School of Economics, Peking University, Beijing, China

*Corresponding Author. Email: sifan@pku.org.cn

Abstract. Indirect prompt injection allows the tool-calling agent to execute calls on untrusted data, and the tool whitelist does not review the source of parameters. This paper introduces Provenance Gate (PGTI), which sequentially performs tool whitelist and parameter source taint checks before the tool execution, allowing the model to still see all tools. In four AgentDojo tasks, 100 attack pairs, and 60 benign tasks, Qwen2.5 14B ran 960 times. The attack success rate decreased from 17.0% with no defense to 3.0%, and it still held after switching models and attack templates; the utility under attack decreased from 40.0% to 28.0%. The benign utility relative to static tool filtering (35.0% and 26.7%) and the increment over the whitelist alone showed no significant difference in this sample. Pair comparison simultaneously reported both safety and utility.

Keywords: indirect prompt injection, LLM agents, tool invocation, information-flow control, AgentDojo

1. Introduction

Tool-calling agents allow Large Language Models (LLMs) to invoke external interfaces [1]. Indirect Prompt Injection (IPI), proposed by Greshake et al. [2], embeds prompts into retrieval data. Deng et al. [3] frame interactions with untrusted external entities as knowledge gaps. Zhan et al. [4] evaluated 30 tool-calling agents using 1,054 test examples, while Yi et al. [5] attributed the success of attacks to the models' inability to distinguish informative context from executable instructions.

There are limitations to each defense mechanism. (1) Static tool filtering removes tools visible to the model, which is an in-built baseline in AgentDojo as proposed by Debenedetti et al. [6]. (2) Tool whitelisting can be derived from user commands [7], but it does not prevent attacks on the same tools. (3) The CaMeL execution capability constraints proposed by Debenedetti et al. [8] solve 77% of AgentDojo tasks, which is lower than 84% for systems without any defense mechanisms.

For this reason, this paper proposes Provenance Gate (PGTI). The research questions are: RQ1 whether PGTI can reduce the Attack Success Rate (ASR); RQ2 whether PGTI can maintain more benign utility

Copyright: © 2026 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

compared with static tool filtering; RQ3 whether argument provenance check can reduce the ASR compared with using only tool whitelists.

The contributions of this paper are: (1) PGTI sequentially determines the tool whitelist and parameter source before tool execution, and the model can still see all tools (Figure 1); (2) we compare the ASR, utility under attack, and benign utility of PGTI with baseline on four sets of tasks (Table 1); (3) we show the attacks that two layers respectively block using ablation experiments (Figure 2).

2. Literature review

2.1. Model layer, prompt layer and detection class defense

Tang et al. [9] comprehensively review the attacks, defenses, and applications of tool-calling agents. The SecAlign [10] mechanism in the model layer fine-tunes the model based on preference optimization, which requires model weights [11]. The prompt layer includes AgentDojo [6], which incorporates a delimiting defense, and DataFilter [11], which removes malicious instructions before data reaches the large language model. In detection-based defenses, Abdelnabi et al. [12] detected task drift from activation differences before and after processing external data, which can incur utility loss [11].

Zhan et al. [13] demonstrated adaptive attacks that bypassed eight types of defenses with a success rate exceeding 50%. Shi et al. [14] introduced malicious tool documents into the tool library with ToolHijacker to manipulate tool selection. The source verification before tool execution remains left to the system-level defense.

2.2. Whitelist of tools and information flow control

System-level defense targets prompts for retrieving data placed by attackers during reasoning [2]. Gong and Deng [7] propose PlanGuard, which only extracts a reference set of effective actions based on user instructions, first blocking unauthorized tool calls with hard constraints, which falls under tool whitelist (allowlist) defense. AgentDojo [6] incorporates a static tool filtering baseline. Jia et al. [15] propose Task Shield, which aligns tasks so that each action serves the user's goals.

Information flow control is based on the lattice model from Denning [16] as the foundation for taint inference. The CaMeL framework by Debenedetti et al. [8] enforces capability constraints at the point of tool invocation, enabling it to complete 77% of AgentDojo tasks with provable security, compared to 84% without a defense system. Jia et al. [17] simultaneously evaluate effectiveness and general utility, finding that the performance of existing defenses is lower than that reported in the original study.

Same-tool attacks reuse the tools of user tasks, where parameters are taken from untrusted data, and the tool whitelist cannot block them. H1 proposes that provenance gates that combine taint checking can reduce the success rate of attacks on tool-calling agents.

3. Methodology

3.1. Threat model and execution cycle

The attacker inserts the prompt into the retrieval data, i.e., indirect prompt injection [2]. The model generates tool calls $c = (t, a)$ based on the user's request q with all tool descriptions, containing the tool name t and parameters a. The provenance gate is located after the call is generated and before the tool execution; the blocked calls are not executed, and the model continues after receiving a policy error message.

3.2. Provenance gate

Provenance Gate (PGTI) is located before the tool execution, sequentially checking the tool whitelist and parameter source (Figure 1), and all tools remain visible to the model.

The tool whitelist (allowlist) is generated by the planner. The planner refers to the isolated planner of Gong and Deng [7], which is a single independent LLM call that only considers the q and the list of tools, without looking at any tool outputs. Tools with prefixes like get_, read_, search_, list_, check_, and find_ belong to reading tools and are not restricted by the whitelist; all other tools that change their state must be within the whitelist.

$$G_1(c) = 1[t \notin A \cup R](1) \tag{1}$$

Among them, $1[\cdot]$ is the indicator function, t is the tool name, A is the tool whitelist, and R is the collection of reading tools.

Regarding the parameter provenance check, its taint inference is based on the information flow model of Denning [16]. The URL, email address, and IBAN regular match values in a, along with sensitive parameters with a length of no more than 80 characters, form an atomic set $V(a)$.

$$\tau(v) = 1[v \notin q \wedge S(v) \neq \varnothing \wedge \forall s \in S(v) : f(s) = 1](2) \tag{2}$$

$$G_2(c) = \tau(v)(3) \tag{3}$$

Among them, v is an atom, $S(v)$ is the set of 400-character segments before and after v in the previous tool output, s is a segment within it, and f is the segment classifier. This classifier is a single independent LLM call that only answers YES or NO; $f(s) = 1$ indicates that s is classified as containing injected instructions.

The call c is executed only when $G_1(c) = 0$ and $G_2(c) = 0$. Calls that are not within $A \cup R$ are blocked by the whitelist; in the remaining calls, those whose $V(a)$ includes tainted atoms are blocked by the taint check.

3.3. Experimental setting

The PGTI configuration above was evaluated using the AgentDojo 0.1.35 benchmark [6] (suite version v1.2.2). The user tasks and injection tasks for banking, Slack, travel, and office suites are sequentially 16 and 9, 21 and 5, 20 and 7, and 40 and 14. Each set was randomly sampled with the seed 20261103 to draw 25 task pairs and 15 benign user tasks, resulting in a total of 100 pairs and 60 benign tasks. The attack template is important_instructions, which does not include model names.

The model is the local Ollama Qwen2.5 14B instruction version (qwen2.5:14b-instruct, RTX 5080 GPU), temperature 0, seed 42, context 12288, a maximum of 1024 tokens generated per instance. The model was selected from non-defense pre-experiments, and there were no defense results during the selection process. 6 configurations include non-defense, AgentDojo's built-in delimiting defense and static tool filtering, using only tool whitelist, using only taint check and PGTI complete configuration. 960 runs use the same batch of samples for 3.4 calculation indicators.

3.4. Evaluation indicators

Attack Success Rate (ASR) is the proportion of attack pairs in which the injected target is completed. Benign utility and utility under attack are the proportion of user tasks completed when there is no injection and when there is an injection. The judgment is based on the AgentDojo judgment function [6], and the proportion is accompanied by a 95% Wilson interval. The configuration differences are tested using the paired exact McNemar test [18], with a significance level of 0.05:

$$p = min(1,\ 2\sum_{i=0}^{min(b,c)}(\frac{b+c}{i})2^{-(b+c)})(4) \tag{4}$$

Among them, b and c are the counts of the two configuration results that are inconsistent in the same case, i is the index for summation, and p is the two-sided p-value.

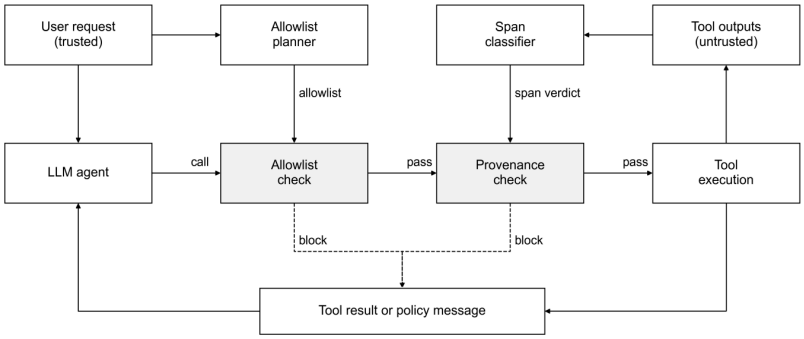


Figure 1. Provenance-gated tool invocation in the agent loop

4. Results

4.1. Attack success rate and utility

The Provenance Gate (PGTI) reduced the attack success rate from 17.0% without defense to 3.0% (inconsistent counts 15 to 1, p = 0.0005, n = 100), H1 was established. Static tool filtering was 5.0%, and this sample did not indicate a difference between it and PGTI (Table 1). At the same time, the utility under attack was reduced from 40.0% to 28.0% (8 to 20, p = 0.036).

For RQ2, the PGTI's benign utility was 35.0%, the static tool filtering was 26.7%, and this sample did not determine any difference between the two. The non-defensive benign utility was 40.0%, and no difference was determined between it and PGTI (inconsistent counts and p-values are shown in Table 1).

When looking at suites separately, the success rate of attacks without defense is concentrated in Slack (36%) and banking (28%), PGTI dropped to 12% and 0%; travel decreased from 4% to 0%, and office was 0%. The benign utility in banking decreased from 60.0% to 46.7%, Slack decreased from 33.3% to 26.7%, and travel and office remained unchanged.

4.2. Ablation comparison

The results of the ablation comparison are shown in Figure 2.

The attack success rate of PGTI is 3.0%. For RQ3, using the tool whitelist alone was 7.0%, and this sample did not show any increment of the taint check; using the taint check alone was 8.0%, and there was no difference detected (see Table 1 for inconsistent counts and p-values).

Out of 17 pairs where the attack succeeded without defense, 6 were blocked by the tool whitelist under PGTI, 8 were blocked by the taint check, and 1 pair that was not blocked did not succeed, while 2 pairs did succeed. Additionally, one pair without defense did not succeed but under PGTI did succeed, so PGTI had a total of 3 succeeding pairs.

The runs with an interception during benign tasks have a PGTI of 3/60, and single-layer configuration is all 2/60. The PGTI single-instance takes 31.3 s, and without defense it takes 19.6 s.

4.3. Robustness test

Comparing no defense and PGTI on the same batch of 100 pairs, when switching to the Qwen2.5 7B model, ASR was 11.0% and 4.0% (p = 0.039), and when switching to the ignore_previous attack template, it was 20.0% and 7.0% (p = 0.011), and H1 remained valid. The utility under attack was 28.0% and 20.0% (p = 0.057) for the former, and this sample did not show a significant difference; the latter was both 30.0% (p = 1.000).

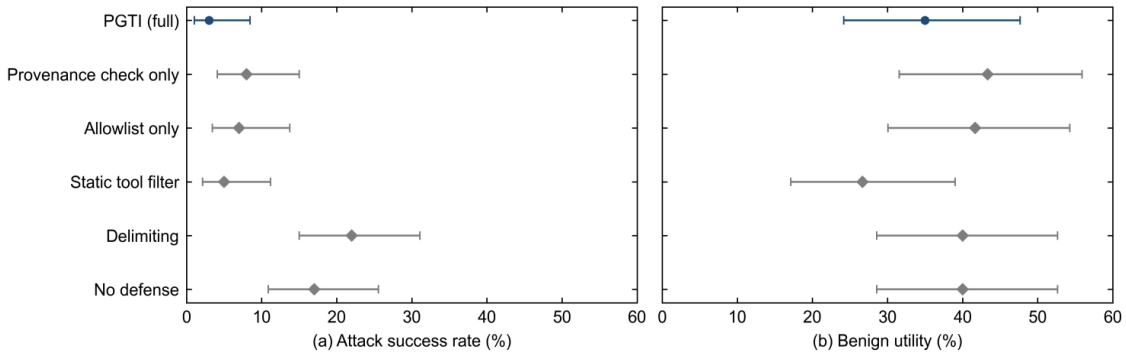


Figure 2. Attack success rate and benign utility with 95% Wilson intervals by configuration

Table 1. Attack success rate, utility and mean run time by configuration (100 attack pairs and 60 benign tasks)

Configuration	ASR (%) [95% CI]	ASR vs PGTI (b : c; p)	Utility under attack (%)	Benign utility (%) [95% CI]	Benign vs PGTI (b : c; p)	Mean time (s)
No defense	17.0 [10.9, 25.6]	15:1; 0.0005	40.0	40.0 [28.6, 52.6]	10:7; 0.629	19.6
Delimiting	22.0 [15.0, 31.1]	20:1; < 0.0001	35.0	40.0 [28.6, 52.6]	9:6; 0.607	21.5
Static tool filter	5.0 [2.1, 11.2]	5:3; 0.727	36.0	26.7 [17.1, 39.0]	7:12; 0.359	20.7
Allowlist only	7.0 [3.4, 13.8]	7:3; 0.344	38.0	41.7 [30.1, 54.3]	7:3; 0.344	32.3
Provenance check only	8.0 [4.1, 15.0]	7:2; 0.180	28.0	43.3 [31.6, 55.9]	10:5; 0.302	34.4
PGTI (full)	3.0 [1.0, 8.5]	-	28.0	35.0 [24.2, 47.6]	-	31.3

5. Discussion

The difference between the two single-layer configurations of ASR 7.0% and 8.0% with PGTI 3.0% was not determined in this sample.

All three PGTI succeeding pairs are in the Slack suite, with two of them running with a taint interception, the attack was completed through other calls, and one of them did not intercept.

The utility under attack decreased from 40.0% to 28.0%, and among the 20 pairs where users completed their tasks without any defense but not under PGTI, 8 pairs had interception and 12 pairs did not; the decrease cannot be fully attributed to interception. Static tools filter ASR 5.0%, and the difference with PGTI is not determined, but PGTI does not remove visible tools of the model [6]. PlanGuard [7] only determines whether the tool is authorized, while CaMeL [8] enforces capability constraints at the point of tool invocation.

The experiment only used one main model and one seed; both sets of robustness settings were local open-source models. Next, we will rerun with multiple models. The attack only used static templates; adaptive attacks were not tested [13]. The segment classifier reads untrusted text, and next, we will design adaptive attacks targeting it. The sample size was 100 pairs and 60 benign tasks; the comparison of RQ2 and RQ3 was

not determined, so we will expand the paired sample and re-test it next. The average single-instance processing time increased from 19.6 s to 31.3 s; the no-defense baseline benign utility was only 40.0%; next, we will measure the cost on models with higher baseline utility.

6. Conclusion

In four tasks of AgentDojo, the provenance gate reduced the attack success rate from 17.0% to 3.0%, and after switching models and attack templates, H1 remained valid, with a decline in utility under attack. This sample did not determine that its benign utility was higher than that of static tool filtering, nor did it determine that the taint check further reduced the attack success rate. Next, we will expand the paired sample and test adaptive attacks.

Declaration of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Author contribution

Lulu Jing: Conceptualization, Methodology, Software, Formal analysis, Writing - original draft, Writing - review & editing. Nanmei Chen: Investigation, Data curation, Writing - review & editing. Zhongmiao Sun: Validation, Visualization, Writing - review & editing. Chao He: Resources, Validation, Writing - review & editing. Fan Si: Supervision, Project administration, Writing - review & editing.

References

[1] Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems, 36*, 68539–68551. https://doi.org/10.52202/075280-2997

[2] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 79–90. https://doi.org/10.1145/3605764.3623985

[3] Deng, Z., Guo, Y., Han, C., Ma, W., Xiong, J., Wen, S., & Xiang, Y. (2025). AI agents under threat: A survey of key security challenges and future pathways. *ACM Computing Surveys, 57*(7), 1–36. https://doi.org/10.1145/3716628

[4] Zhan, Q., Liang, Z., Ying, Z., & Kang, D. (2024). InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. *Findings of the Association for Computational Linguistics: ACL 2024*, 10471–10506. https://doi.org/10.18653/v1/2024.findings-acl.624

[5] Yi, J., Xie, Y., Zhu, B., Kiciman, E., Sun, G., Xie, X., & Wu, F. (2025). Benchmarking and defending against indirect prompt injection attacks on large language models. *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, 1809–1820. https://doi.org/10.1145/3690624.3709179

[6] Debenedetti, E., Zhang, J., Balunovic, M., Beurer-Kellner, L., Fischer, M., & Tramer, F. (2024). AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *Advances in Neural Information Processing Systems, 37*, 82895–82920. https://doi.org/10.52202/079017-2636

[7] Gong, G., & Deng, Z. (2026). PlanGuard: Defending agents against indirect prompt injection via planning-based consistency verification. *2026 International Joint Conference on Neural Networks*, 1–6.

https://doi.org/10.1109/ijcnn69854.2026.11728563

[8] Debenedetti, E., Shumailov, I., Fan, T., Hayes, J., Carlini, N., Fabian, D., Kern, C., Shi, C., Terzis, A., & Tramer, F. (2026). Defeating prompt injections by design. *2026 IEEE Conference on Secure and Trustworthy Machine Learning*, 587–618. https://doi.org/10.1109/satml68715.2026.00040

[9] Tang, Y., Liu, Y., Lan, J., Yan, Z., & Gelenbe, E. (2026). Security of LLM-based agents regarding attacks, defenses, and applications: A comprehensive survey. *Information Fusion, 127*, 103941. https://doi.org/10.1016/j.inffus.2025.103941

[10] Chen, S., Zharmagambetov, A., Mahloujifar, S., Chaudhuri, K., Wagner, D., & Guo, C. (2025). SecAlign: Defending against prompt injection with preference optimization. *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, 2833–2847. https://doi.org/10.1145/3719027.3744836

[11] Wang, Y., Chen, S., Alkhudair, R., Alomair, B., & Wagner, D. (2026). Defending against prompt injection with DataFilter. *2026 IEEE Conference on Secure and Trustworthy Machine Learning*, 139–157. https://doi.org/10.1109/satml68715.2026.00017

[12] Abdelnabi, S., Fay, A., Cherubin, G., Salem, A., Fritz, M., & Paverd, A. (2025). Get my drift? Catching LLM task drift with activation deltas. *2025 IEEE Conference on Secure and Trustworthy Machine Learning*, 43–67. https://doi.org/10.1109/satml64287.2025.00011

[13] Zhan, Q., Fang, R., Panchal, H. S., & Kang, D. (2025). Adaptive attacks break defenses against indirect prompt injection attacks on LLM agents. *Findings of the Association for Computational Linguistics: NAACL 2025*, 7101–7117. https://doi.org/10.18653/v1/2025.findings-naacl.395

[14] Shi, J., Yuan, Z., Tie, G., Zhou, P., Gong, N., & Sun, L. (2026). Prompt injection attack to tool selection in LLM agents. *Proceedings 2026 Network and Distributed System Security Symposium*. https://doi.org/10.14722/ndss.2026.230675

[15] Jia, F., Wu, T., Qin, X., & Squicciarini, A. (2025). The task shield: Enforcing task alignment to defend against indirect prompt injection in LLM agents. *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 29680–29697. https://doi.org/10.18653/v1/2025.acl-long.1435

[16] Denning, D. E. (1976). A lattice model of secure information flow. *Communications of the ACM, 19*(5), 236–243. https://doi.org/10.1145/360051.360056

[17] Jia, Y., Shao, Z., Liu, Y., Jia, J., Song, D., & Gong, N. (2026). A critical evaluation of defenses against prompt injection attacks. *Proceedings of the 31st ACM Symposium on Access Control Models and Technologies*, 265–270. https://doi.org/10.1145/3750555.3811884

[18] McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika, 12*(2), 153–157. https://doi.org/10.1007/bf02295996