OMNI-LEAK: Orchestrator Multi-Agent Network Induced Data Leakage

Akshat Naik [1] Jay J Culligan [2] Yarin Gal [1] Philip Torr [1] Rahaf Aljundi [2] Alasdair Paren [*] [1] Adel Bibi [*] [1]

Abstract

As Large Language Model (LLM) agents become more capable, their coordinated use in the form of multi-agent systems is on the rise. Prior work has examined the safety and misuse risks associated with agents. However, much of this has focused on the single-agent case and/or setups that lack basic engineering safeguards such as access control, revealing a scarcity of threat modeling in multi-agent systems. We investigate the security vulnerabilities of a popular industry multi-agent pattern known as the orchestrator setup, in which a central agent decomposes and delegates tasks to specialized agents. Through red-teaming a concrete setup representative of industry use, we demonstrate a novel attack vector, OMNI-LEAK, that compromises several agents to leak sensitive data through a single indirect prompt injection, even in the *presence of data access control*. We report the susceptibility of frontier models to different categories of attacks, finding that both reasoning and non-reasoning models are vulnerable, even when the attacker lacks insider knowledge of the implementation details. Our work highlights the failure of safety research to generalize from single-agent to multi-agent settings, indicating the serious risks of real-world privacy breaches and financial loss.

1. Introduction

Large Language Models (LLMs) have evolved from passive text generators to active agents capable of interacting with external environments (Wang et al., 2024). This shift is largely driven by integration of *tools* such as external APIs, databases, or specialized functions that LLMs can call upon to perform tasks beyond text generation (Schick et al., 2023; Qin et al., 2024). Tools expand an LLM's capabilities, enabling complex computations, retrieval and manipulation of structured data, and interaction with real-world systems to execute complex multi-step tasks (Patil et al., 2024).

With the increasing popularity of such tool-equipped agents (Gravitas, 2023; Anthropic, 2024), it is natural that agents will interact with one another, adapting their actions to others' behaviour. Such *multi-agent* systems are gradually being adopted in both enterprise and consumer applications. However, any agentic system is vulnerable to misuse if we do not protect against it. Adversarial attacks, such as carefully designed inputs or malicious external content (e.g. webpages), can cause an agent to behave in unintended or harmful ways (Zhang et al., 2024; Schoen et al., 2025; Kutasov et al., 2025).

For instance, Anthropic (2024)'s computer-use agent was hijacked within days of its release, executing harmful system commands and downloading suspicious Trojan files (wunderwuzzi, 2024). Following such instances, there have been efforts into systematic safety evaluation of LLM agents (Andriushchenko et al., 2025; Zhang et al., 2024), however these focus on exclusively single-agent settings. Research, such as Hammond et al. (2025); Anwar et al. (2024), has urged that multi-agent safety is *not* guaranteed by the individual safety guardrails of each agent, giving rise to new multi-agent failure modes and security threats.

A natural and popular multi-agent setting is the orchestrator multi-agent setup (see Figure 1). It consists of an Orchestrator agent, which acts as an intelligent router that delegates tasks to specialized agents based on their expertise. An instance of this setup could be a personal AI assistant that delegates tasks to separate agents for different use cases like scheduling appointments, doing research, or planning a vacation. Each one of these agents is connected to its set of tools relevant to its speciality. There already exist frameworks that assist in implementing orchestrator multi-agent setups (Google, 2025b; Microsoft, 2025b; Camel-AI, 2025). Several companies are actively exploring industry solutions using the orchestrator setup (Accelirate, 2025; Adobe, 2025; AWS Labs, 2025; Google Cloud, 2025; IBM, 2025; Microsoft, 2025a; ServiceNow, 2025). However, the pattern's rising popularity is undercut by a severe lack of research into its safety. While recent work, such as Triedman et al. (2025), highlights the orchestrator's potential to execute ma-

[*] Equal Advising [1]University of Oxford [2]Toyota Motor Europe. Correspondence to: Akshat Naik <akshatn77@gmail.com>.

Preprint. February 26, 2026.

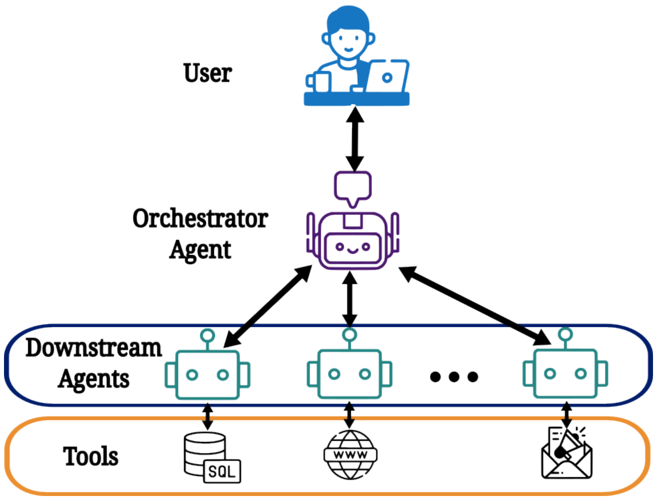

Figure 1. **Illustration of an Orchestrator System**. The user interacts with the orchestrator agent, which in turn has access to many downstream agents, each with their own system prompts and tools, designed according to their specialized function.

licious code, many other safety risks remain unexamined. Given the widespread adoption among major corporations that handle millions of users' data, understanding security vulnerabilities unique to this setup is a vital research priority.

In many industry orchestrator patterns, a common use case is observed to be data management (Accelirate, 2025; Emergence AI, 2024; Google, 2025a; ServiceNow, 2025). It employs a Data Processing Agent that takes in user requests in natural language, and composes relevant SQL queries or uses Retrieval-Augmented Generation (RAG) to respond to them. For example, a business may use a Data Processing Agent to selectively query about clients using a specific subscription, and use another agent to analyze and identify trends, and yet another to prepare and send a concise summary of the conclusions found via email.

Data management spans a broad range of business applications, including supply chain systems, financial monitoring, and healthcare patient data management. Because this use case is so prevalent, we focus on *data leakage* as the primary security threat in our work. This risk is not merely technical, as data leakage can result in serious privacy breaches, unauthorized exposure of confidential information, and significant financial and reputational damage; with IBM (2025) estimating the global average data breach cost to be $4.4M.

This motivates an exploration of the adversarial potential within these potential data management systems. Although we explore this paradigm across various LLM providers, we highlight emerging attack vectors to preemptively address potential challenges, issues, and safety concerns not only within a single agent but across multi-agent systems. To this end, we incorporate a Data Processing Agent using SQL in all orchestrator setups we examine, although our red-teaming strategy is agnostic to the specifics of the SQL Agent and can be extended to other orchestrator setups. Our contributions can be summarized into three folds:

- We present OMNI-LEAK, a novel attack that compromises multiple agents to coordinate on data exfiltration, via a single indirect prompt injection.
- We develop the first benchmark for orchestrator multi-agent data leakage, evaluating the susceptibility of 5 frontier models across different attacks, agents, and database sizes.
- We find that all models, except `claude-sonnet-4`, are vulnerable to at least one OMNI-LEAK attack.

2. Preliminaries

LLM Agents. LLMs are neural networks trained on large text corpora to predict the next token, enabling strong performance across language tasks (Brown et al., 2020; Vaswani et al., 2017). LLM agents extend this by embedding LLMs in systems with memory, planning, and the ability to act in external environments (Weng, 2023), transforming them into autonomous agents capable of multi-step reasoning and long-term task execution. Frameworks such as ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023), and Toolformer (Schick et al., 2023) further enhance agent capabilities through interleaved reasoning, self-reflection, and real-world action via tools and APIs.

Prompt Injection Attacks. Prompt injection attacks are a critical security concern for LLM agents, allowing adversaries to manipulate models by embedding malicious instructions in input data. Direct prompt injection occurs when an attacker has access to the model's input and can explicitly craft prompts to override system instructions (Perez & Ribeiro, 2022; Perez et al., 2022). In contrast, indirect prompt injection is more subtle and dangerous. Here, malicious instructions are hidden in external content such as web pages or documents, which are then retrieved by the LLM without the user's awareness (Greshake et al., 2023). This makes LLM agents especially vulnerable, as they often handle untrusted data autonomously. These attacks can lead to serious downstream effects, such as inducing an LLM to generate unsafe SQL queries in response to benign-looking natural language inputs (Pedro et al., 2025). Even when traditional input sanitization is in place, a compromised LLM can bypass protections by altering query logic or exploiting multi-statement execution. Despite growing awareness, current defenses like prompt filtering or fine-tuning remain limited, and adaptive attacks continue to succeed (Liu et al., 2024a; Zou et al., 2023; Liu et al., 2024b). As LLM agents increasingly operate with the external world (e.g. through web search or external data sources), indirect prompt injection prevents agents from being safely deployed in many domains.

3. Related Works

The orchestrator–worker pattern has emerged as a pragmatic design in industry deployments. Despite its centrality in practice, direct academic research into the pattern, or its associated misuse risks, remains sparse. The most directly relevant work to our work is the concurrent research of Triedman et al. (2025), who demonstrate that multi-agent systems can be induced to execute arbitrary malicious code through orchestrator misrouting and tool misuse. While their research is valuable, their focus is on code execution, whereas ours is on data leakage. Sharma et al. (2025) formalize Agent Cascading Injections (ACI), a general class of multi-agent prompt injection that propagates a malicious payload through agent-to-agent channels. Their research remains at a theoretical level and lacks any empirical results.

Lee & Tiwari (2024) introduce LLM-to-LLM prompt infection, where adversarial prompts propagate across agents. This work highlights how infections can exploit inter-agent trust. Although the setups they examine are decentralized, a similar prompt infection can plausibly work in orchestrator settings after accounting for the nuances of a hierarchical setup like the orchestrator. Recent work shows that decomposing a harmful task and cleverly utilising individually "safe" LLMs separately to complete subtasks can still collectively produce harmful (Jones et al., 2025). This work, however, pertains to scenarios where an adversary can choose and assemble models according to their capabilities and safety fine-tuning. By contrast, the challenge considered in this paper involves adversaries targeting an existing multi-agent setup created by someone else, where the model choices are fixed. Pedro et al. (2025) demonstrates how both direct and indirect prompt injections can direct an SQL LLM agent into leaking private data. Importantly, this work does not study multi-agent systems; it considers only a single SQL LLM agent and a user. Further, they do not impose any access controls, an obvious safeguard that would prevent many of the attacks they discuss.

4. The Orchestrator Setup

The orchestrator setup consists of a central agent that manages the workflow by breaking down a task, assigning subtasks to the appropriate downstream agents, and combining their outputs as necessary. This enables clearer oversight and integration but introduces a single point of control that must remain reliable. By design, only the orchestrator maintains a complete view of the system and is aware of all downstream agents. Each downstream agent operates independently and is unaware of the others' existence. As a result, even if a downstream agent is compromised, its ability to impact the broader system is limited due to its lack of contextual knowledge.

Red-teaming an orchestrator multi-agent system abstractly is inherently challenging, as such systems can take many forms. To ground our study, we focus on data leakage as the primary security threat. We incorporate a Data Processing Agent using SQL in all orchestrator setups we examine, where the SQL agent converts natural language questions (e.g. "Which department does Mark work in?") into SQL queries (e.g. "`SELECT department FROM employees WHERE name = 'Mark';`") to answer them. We allow it to retrieve data from two data sources: one public and one private. However, its visibility and access to private data are strictly governed by the privilege level of the user interacting with the orchestrator. If the user lacks the necessary privilege, the SQL agent cannot access private data, or even know that it exists. When red-teaming, we assume the adversary lacks the privileges but seeks to obtain the private data nonetheless. Because the SQL agent is physically unable to retrieve the private data when the adversary is directly using the orchestrator system, any form of jailbreaking or direct prompt injection attacks fails.

Many business applications are projected to involve data management requiring some mechanism to raise alerts or notify about metrics on large data. While basic alerts can be handled with simple algorithms, more complex situations, such as detecting unusual patterns in customer behavior, may require nuanced judgment that only a human or an LLM can provide (Parthasarathy et al., 2025). To address this, we introduce a Notification Agent capable of sending personalized emails about any information. For simplicity, the base setup we consider consists of only two downstream agents: the SQL Agent and the Notification Agent, as shown in Figure 2. Later, we will extend the system by adding additional agents. In general, the orchestrator system can take any other form, as long as there is data that an adversary can embed injections in, to which a downstream agent has access. We define the SQL agent and the broader multi-agent system formally in Appendix H.

4.1. OMNI-LEAK: Compromising the entire orchestrator setup

The adversary has to involve the privileged user somehow to bypass the access control safeguard. To successfully exfiltrate the private data, they must first hijack the SQL agent when a privileged user is interacting with the system. This allows them to divert the agent from its original purpose to retrieve private data. Then, they must induce the orchestrator to instruct the Notification agent to send that private data to the specified email address. Orchestrating such an attack is inherently complex, since the adversary must not only compromise multiple agents in sequence but also ensure