# Your Wish Is My Tool Call: A Security Analysis of WeChat's Mini-Program AI Integration Toolkit

Zidong Zhang
Simon Fraser University
Canada
zza323@sfu.ca

Wenrui Diao
Shandong University
China
diaowenrui@link.cuhk.edu.hk

Jianliang Wu
Simon Fraser University
Canada
wujl@sfu.ca

## Abstract

WeChat's mini-program AI toolkit automates the conversion of application functions into atomic interfaces that an in-app large language model agent can select and invoke. This changes the execution boundary: business content becomes model input, while model-selected operations execute in the user's authenticated context. We manually analyze the toolkit's generation guidance, code templates, interface restrictions, and validator and characterize four security concerns in this workflow: untrusted content can influence subsequent interface calls; model inference can be treated as user authorization; schema and server-domain checks do not provide business authorization; and structural validation does not establish runtime security. We ground the analysis in a case study of WeStoreCafe, WeChat's official reference mini-program. From its source code, we reconstruct an internal path through local mini-program state and a controlled external path through a shared backend, connecting the four problems to catalog, order, and address operations.

## CCS Concepts

• **Security and privacy → Software and application security**.

## Keywords

Mini-program Security; LLM Agent Security;

**ACM Reference Format:**
Zidong Zhang, Wenrui Diao, and Jianliang Wu. 2026. Your Wish Is My Tool Call: A Security Analysis of WeChat's Mini-Program AI Integration Toolkit. In *4th Workshop on Security of Agentic and Miniapp Empowered Mobile Apps (SaTS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3846386.3848859

## 1 Introduction

Super-apps have transformed mini-program ecosystems into tool libraries for in-app AI agents. For example, WeChat's mini-program AI specification [15] allows mini-programs to expose "atomic interfaces" and "atomic components" that a large language model (LLM) can invoke on behalf of users; JD, Meituan, and Didi were onboarded within weeks. This shift matters not merely as a technical integration, but because it operates at super-app scale: monthly active mini-program audiences reached 973 million on WeChat, 644 million on Alipay, and 273 million on Douyin [2]. Beyond mini-program ecosystems, the same paradigm appears in the broader tool-use and function-calling architectures underlying the Model Context Protocol (MCP) [6], OpenAI GPT Actions, and Microsoft 365 Copilot [5].

In-app AI agents change how mini-program actions are selected and parameterized. Without agent integration, a user reaches a business function through the mini-program's visible UI: the selected UI element invokes a developer-defined handler, and application code maps UI inputs and application state to its parameters. With agent integration, the mini-program additionally exposes business functions as atomic interfaces. The agent selects an interface, while the model constructs its arguments from the conversation and prior tool results. The selected interface then executes inside the mini-program and may access business backends or native JSAPIs under the user's authenticated session.

Developers can implement and register these skills following WeChat's documentation [15], or use its official toolkit to generate skill packages, validate their implementation, and evaluate simulated tasks [11]. WeChat also provides WeStoreCafe as a reference ordering application with an implemented skill [10].

However, combining model-influenced inputs with execution under an authenticated session introduces additional security risks. First, third-party content returned by one atomic interface may be interpreted as instructions for the next operation, enabling indirect prompt injection [3, 8]. Second, the agent may invoke an atomic interface that the user did not request. For example, it may cancel an order or save an address when the user only asked for information. Third, a mini-program can pass the platform's checks while its backend still fails to verify whether the current user can access the requested data. This risk extends identity and request-authorization problems previously documented in mini-program ecosystems [4, 16–18]. Finally, passing WeChat's official validator only reflects the structural correctness and execution compatibility, and does not mean that a deployed call reflects user intent or a backend enforces object ownership.

**Our Work.** We manually analyze WeChat's official toolkit for automating mini-program AI integration, including its generation guidance, code templates, and validator. We trace how toolkit-generated skills expose business content to the model, translate model output into atomic-interface calls, and execute those calls in the user's authenticated context. We trace four security concerns through this workflow and show how general agent risks, backend authorization, and development-time assurance meet at the atomic interface boundary.

We then present a case study of WeStoreCafe, WeChat's official reference mini-program. From its source code, we reconstruct how it exposes catalog, order, and address operations to the agent. We examine both its internal path, where registered handlers operate



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
*SaTS '26, The Hague, Netherlands*
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-3018-4/2026/11
https://doi.org/10.1145/3846386.3848859

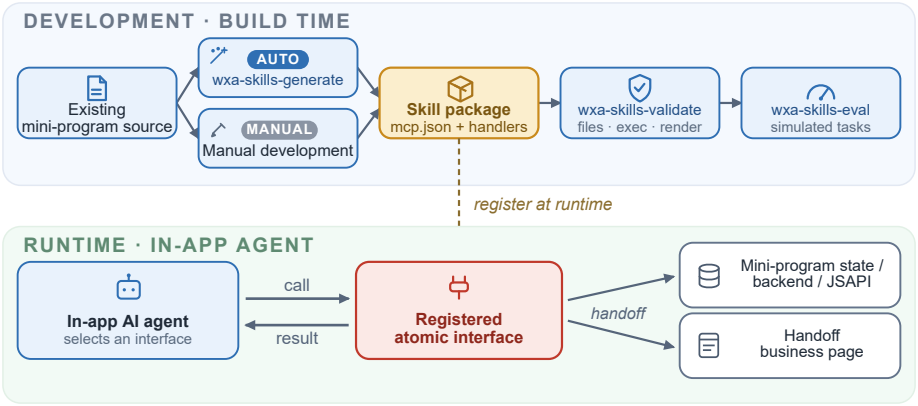



**Figure 1: WeChat mini-program AI development and runtime workflow.**

on local mini-program state, and a controlled external path, where the same business model is connected to a shared backend. The case study shows how the four concerns map to concrete interfaces and business consequences.

**Contributions.** This work makes three contributions:

- We provide a source-level security analysis of WeChat's official AI integration toolkit and characterize the trust boundaries introduced by toolkit-generated atomic interfaces.
- We discuss how four established concerns, indirect prompt injection, intent mismatch, backend object authorization, and validation scope, intersect in the official mini-program AI workflow.
- We present a case study of the official WeStoreCafe reference mini-program, tracing its internal execution path and a controlled external backend path to concrete security consequences.

## 2 Background

**Mini-Program AI Skills.** A WeChat mini-program skill is an independent subpackage that makes selected application functions available to the in-app AI agent. Its `mcp.json` file lists each atomic interface's name, natural-language description, `inputSchema`, `outputSchema`, and optional component or page path. The skill creates an instance with `wx.modelContext.createSkill` and registers a JavaScript handler for each declared interface through `registerAPI` [15]. When the agent selects an interface, its handler receives arguments supplied by the model and executes inside the mini-program. A handler can return `content` for the textual response and `structuredContent` that the agent can use in later calls. An optional atomic component can render this structured result. Alternatively, a `handoff` result can present a mini-program card that opens an ordinary application page.

**Development Toolkit.** The toolkit provides three tools: `wxa-skills-generate` creates skill packages from existing mini-programs; `wxa-skills-validate` checks their structure, interface execution, and component rendering; and `wxa-skills-eval` evaluates simulated conversations, including intent understanding, call sequences, and final responses [12–14]. Figure 1 places these development tools alongside the runtime workflow. We use *validator* below to refer to `wxa-skills-validate`.

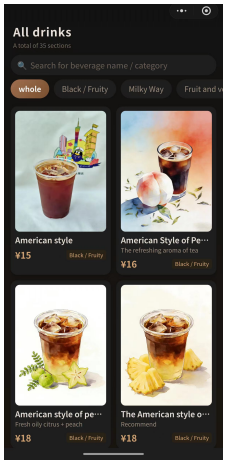



**Figure 2: The index page of WeStoreCafe Mini-program.**

**WeStoreCafe Mini-program.** WeStoreCafe is WeChat's official reference and onboarding demo for the supported integration workflow, rather than a security-hardened or best-practice implementation. It covers browsing drinks, selecting options, creating and managing an order, and entering a delivery address [10]. Figure 2 shows its index page for browsing drinks. The current demo uses AI Handoff: the assistant returns text and a mini-program card, and the user's tap opens a regular business page where the later ordering steps are completed. Thus, the visible conversation path in this demo is a handoff to a normal mini-program page, rather than an in-chat card-rendering workflow.

## 3 Threat Model

**Scope and Settings.** We focus on WeChat mini-programs because WeChat provides public documentation, an official AI integration toolkit, and a reference application that can be analyzed together. For the AI-enabled mini-programs considered in this study, we assume that the integration is implemented through this toolkit. This defines our study population; it is not a claim about every future deployment. We assume that platform login and network restrictions operate as documented; our analysis concerns protections introduced or omitted by the toolkit-generated integration.

**Attacker Assumption and Goals.** The attacker is an ordinary mini-program user with no platform, developer, backend, network, or victim-session privileges. The attacker can use normal application functions and send text to the in-app AI agent, but cannot directly select the interface or arguments ultimately chosen by the model. In the content-mediated scenario, when an application lets ordinary users publish content later read by another user's agent, the attacker seeks to induce an operation that the victim did not request, such as canceling the victim's active order or changing a saved address (so that the order will be delivered to a different address). Separately, in the object-authorization scenario, the attacker supplies a known object identifier through their own authenticated session and seeks to access another user's data when the backend omits an ownership check. We treat the model as a potentially mistaken or manipulated component, not an adversary, and assume non-malicious developers who may nevertheless omit necessary security checks.

## 4 Security Analysis of WeChat's Mini-Program AI Toolkit

**Research Targets.** We examine WeChat's official AI integration toolkit, `ai-mode-skills` v0.2.1, and its reference application, WeStoreCafe, using the repository snapshots accessed on July 22, 2026 [10, 11]. Our manual review covers the instructions and templates in `wxa-skills-generate`, the code and rules in `wxa-skills-validate`, and the demo's eight atomic interfaces.

### 4.1 Risk Sources and Validation Scope

Using the generation templates and demo code, we trace each interface from its `mcp.json` declaration and registration to its handler inputs, returned content, and state changes. Along this path, we examine how the integration handles untrusted content, user intent, and backend authorization, and whether the validator checks these properties. These checks identify the following four concerns, showing how established agent and application-security risks meet in the official integration workflow.

**P1: Untrusted content becomes control input.** When an AI assistant searches for products in a mini-program, it reads the product descriptions returned by the application. If an attacker can modify these descriptions, they may insert instructions into the text. The assistant may follow those instructions and perform an action the user did not request, such as canceling an order. This creates a risk of indirect prompt injection.

**P2: Actions without a user request.** A user may ask an AI assistant to check an order without asking it to cancel the order. The toolkit uses written descriptions to tell the assistant when to call each function, but the assistant may still misunderstand the request. If it chooses to cancel the order and the application executes that choice without checking that cancellation was requested, the order may be canceled against the user's wishes. The application therefore needs its own check that the user requested the action.

**P3: Access to other users' orders.** When an AI assistant requests an order from a server, the server needs to check whether the logged-in user is allowed to view that order. Passing platform checks does not answer this question. If the server returns orders based only on their numbers, a user who knows someone else's order number may obtain that person's order details and address. This access-control failure can also occur when a user makes the request directly, without an AI assistant.

```
const items = picked.map(d => ({
  drinkId: d.id,
  name: d.name,
  price: d.price,
  description: d.description
}))
return {
  structuredContent: { items },
  handoff: { query, payload }
}
```

**Listing 1: Abridged code showing that product descriptions are included in search results returned to the assistant.**

**P4: Limits of toolkit validation.** Developers use the toolkit's validator to check whether functions are correctly connected to the mini-program and whether they can run and display their results. In the version we examined, the rules do not check whether an operation was requested by the user or whether the user may access a particular order. An application can therefore pass these checks while still lacking the protections discussed above.

### 4.2 Case Study of WeStoreCafe

We examine WeStoreCafe, WeChat's official demonstration mini-program for AI-assisted ordering [10]. It gives an AI assistant access to functions for finding drinks, managing orders, and saving addresses. We inspect these functions to understand whether the application checks that an action reflects the user's request. We also use a separate test server to examine how access to orders must be restricted when data is shared across users.

**Actions without a user request.** When the assistant searches for drinks, the application includes product descriptions in the results it returns, as shown in Listing 1. The assistant can also call a function that cancels the current order. The description of this function tells the assistant to call it only when the user requests cancellation. However, the cancellation function itself does not independently check for such a request or require user confirmation before changing the order. Once called, it clears the active order and marks it as canceled.

This matters because the text the assistant reads may influence what it does next. Consider an application that allows users to publish product descriptions that other users' assistants can read. An attacker could insert an instruction to cancel an order into one of these descriptions. If the assistant followed that instruction during a product search, it could cancel the user's order even though the user had only asked to find a drink. This would combine the risk of following untrusted content (P1) with the absence of an independent check on the user's request (P2).

Our analysis does not establish that this attack succeeds in the official demo. Its product descriptions are supplied by the developer, and its orders are stored locally on the user's device. We have not established a way for an ordinary user to modify descriptions read by another user's assistant. Our code review confirms that descriptions are returned to the assistant and that the cancellation

```
const order = orders.get(requestedOrderId)
if (!order) return send(res, 404, { error: 'not found' })
// Missing: order.openid === whoami(req)
return send(res, 200, { order })
```

**Listing 2: Order lookup in our test server, without checking that the order belongs to the requesting user.**

function changes the order. We did not test whether the assistant would follow an instruction inserted into a description.

**Access to other users' orders.** Keeping orders on a shared server raises a separate question: which orders may each logged-in user access? The official demo's local storage does not provide a shared collection of different users' orders in which to examine this question. We therefore built a test server for the same ordering functions. In the example implementation shown in Listing 2, the server retrieves an order using the supplied order number without checking whether the order belongs to the requesting user. Under this condition, a user who knows another user's order number can retrieve that order, including its contents and saved address.

The missing ownership check is in our test implementation. We use this example to illustrate the need to restrict access to each user's orders (P3), and do not claim to have found this flaw in an official WeStoreCafe server. This problem does not require the assistant to follow malicious text. A user who knows another user's order number could make the same request directly. Identifying the logged-in user is therefore insufficient unless the server also checks whether that user is allowed to access the requested order.

**Limits of toolkit validation.** Finally, we examined whether the toolkit's validator checks for these missing protections (P4). Its checks cover whether declared functions have matching implementations, whether returned fields match their declarations, and whether the mini-program can execute and display results. We found no rule requiring the cancellation function to obtain user confirmation or checking that a requested order belongs to the user. Passing these checks therefore does not show that a user requested an operation or is allowed to access a particular order.

## 5 Discussion

**Mitigations.** The four concerns require controls at different layers. Skills should preserve content provenance and separate plans from untrusted tool results; planned tool dependencies can constrain content-driven calls [1]. Policies over tool names and arguments can enforce least privilege and require approval for privilege expansion [9]. Sensitive calls should additionally require runtime confirmation bound to the specific operation and arguments, rather than rely on a natural-language interface description. Security-oriented validator extensions could flag missing confirmation and untrusted flows. These measures cannot replace backend authorization: every endpoint must derive the principal from the authenticated session and check access to each referenced object and business-state transition [7].

**Limitations and Future Work.** This paper reports work in progress based on one version of the WeChat toolkit and one official reference mini-program. WeStoreCafe stores its state locally, and our controlled external path is designed to expose the authorization decisions that arise when the same business model is moved to a shared backend. It is not an official or production WeStoreCafe implementation. We therefore do not claim that every generated skill is vulnerable or that every model will complete the described content-to-action chain. Future work will evaluate the complete chain with the live agent, measure its behavior across repeated trials, analyze a larger set of generated and deployed skills, and assess whether provenance tracking, confirmation policies, and security-oriented validator checks prevent the identified concerns. Public deployment announcements do not expose sufficient skill or backend artifacts for source-level reproduction; broader ecosystem prevalence therefore remains outside this study.

**Ethical Consideration.** All experiments use our own accounts, test data, and controlled services. We do not access third-party data or production systems,and we report conditional consequences separately from runtime-confirmed attacks.

## 6 Related Work

Prior work has established that retrieved third-party content can influence later agent actions, a problem commonly called indirect prompt injection [3]. The OWASP LLM Top 10 also identifies prompt injection and excessive agency as major risks in LLM-integrated applications [8]. Recent agent defenses constrain tool invocation or planned tool dependencies [1, 9]. These works study general agent runtimes; they do not examine a mini-program skill package whose handlers execute inside a logged-in application and whose results can supply context for later calls.

A separate line of work studies security failures in mini-program and super-app ecosystems, including identity confusion, cross-mini-program request forgery, and insufficient separation between application resources [4, 16–18]. These attacks mainly target developer-controlled request paths, where the application fixes the reachable operation and its parameters. Our setting adds an agent layer that selects both the registered function and its arguments from conversation and earlier interface results.

Our work does not claim a new vulnerability category. It connects these two lines of work by examining how established agent and application-security concerns meet in WeChat's official mini-program AI workflow.

## 7 Conclusion

WeChat's AI integration turns model-visible text and model-selected arguments into inputs to mini-program business operations. Our manual analysis of the official toolkit characterizes how four established concerns meet in this workflow: untrusted content can influence control flow, model inference can be treated as user authorization, platform checks do not provide business authorization, and structural validation does not establish runtime security. The WeStoreCafe case study shows how these concerns appear across an internal mini-program path and a controlled external backend path. Because this is a work-in-progress study, establishing ecosystem prevalence, end-to-end exploitability, and mitigation effectiveness remains future work.

## References


[1] Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. 2025. IPIGuard: A Novel Tool Dependency Graph-Based Defense Against Indirect Prompt Injection in LLM Agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Suzhou, China, 1023–1039. https://doi.org/10.18653/v1/2025.emnlp-main.53

[2] CNBC. 2026. Chinese Tech Giants Enter the 'Agentic Commerce' Race as AI Reshapes Super-Apps. https://www.cnbc.com/2026/01/21/china-tech-ai-agentic-commerce-super-apps-alibaba-taobao-qwen-tencent-wechat-doubbao-weixin.html. Accessed: 2026-07-21.

[3] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec)*. arXiv:2302.12173.

[4] Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang Wang. 2020. Demystifying Resource Management Risks in Emerging Mobile App-in-App Ecosystems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[5] Microsoft. 2025. Agents for Microsoft 365 Copilot. Microsoft Learn, https://learn.microsoft.com/en-us/microsoft-365/copilot/extensibility/agents-overview. Accessed: 2026-07-21.

[6] Model Context Protocol Contributors. 2025. Model Context Protocol (MCP) Specification, Version 2025-11-25. https://modelcontextprotocol.io/specification/2025-11-25. Accessed: 2026-07-21.

[7] OWASP Foundation. 2023. API1:2023 Broken Object Level Authorization. OWASP API Security Top 10, https://api-security.owasp.org/editions/2023/en/0xa1-broken-object-level-authorization/. Accessed: 2026-09-15.

[8] OWASP GenAI Security Project. 2025. OWASP Top 10 for Large Language Model Applications (2025): LLM01 Prompt Injection. https://owasp.org/www-project-top-10-for-large-language-model-applications/. Accessed: 2026-07-21.

[9] Tianneng Shi, Jingxuan He, Zhun Wang, Linyu Wu, Hongwei Li, Wenbo Guo, and Dawn Song. 2025. Progent: Programmable Privilege Control for LLM Agents. arXiv preprint. https://arxiv.org/abs/2504.11703 arXiv:2504.11703.

[10] WeChat Mini Program. 2026. AI Mode Demo: WeStoreCafe Ordering Scenario. https://github.com/wechat-miniprogram/ai-mode-demo/tree/240d618. Accessed: 2026-07-22.

[11] WeChat Mini Program. 2026. AI Mode Skills Toolkit for Mini-programs. https://github.com/wechat-miniprogram/ai-mode-skills/tree/0342b55. Accessed: 2026-07-22.

[12] WeChat Mini Program. 2026. AI Mode Skills Toolkit: wxa-skills-eval. https://github.com/wechat-miniprogram/ai-mode-skills/tree/0342b55/wxa-skills-eval. Accessed: 2026-07-22.

[13] WeChat Mini Program. 2026. AI Mode Skills Toolkit: wxa-skills-generate. https://github.com/wechat-miniprogram/ai-mode-skills/tree/0342b55/wxa-skills-generate. Accessed: 2026-07-22.

[14] WeChat Mini Program. 2026. AI Mode Skills Toolkit: wxa-skills-validate. https://github.com/wechat-miniprogram/ai-mode-skills/tree/0342b55/wxa-skills-validate. Accessed: 2026-07-22.

[15] WeChat (Tencent). 2026. WeChat Mini-Program AI Development Mode: Model Context API Reference (`wx.modelContext`, `createSkill`, `registerAPI`). https://developers.weixin.qq.com/miniprogram/dev/ai/reference/api.html. Accessed: 2026-07-21.

[16] Yuqing Yang, Yue Zhang, and Zhiqiang Lin. 2022. Cross Miniapp Request Forgery: Root Causes, Attacks, and Vulnerability Detection. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[17] Lei Zhang, Zhibo Zhang, Ancong Liu, Yinzhi Cao, Xiaohan Zhang, Yanjun Chen, Yuan Zhang, Guangliang Yang, and Min Yang. 2022. Identity Confusion in WebView-based Mobile App-in-app Ecosystems. In *Proceedings of the 31st USENIX Security Symposium (USENIX Security)*.

[18] Zidong Zhang, Qinsheng Hou, Lingyun Ying, Wenrui Diao, Yacong Gu, Rui Li, Shanqing Guo, and Haixin Duan. 2024. MiniCAT: Understanding and Detecting Cross-Page Request Forgery Vulnerabilities in Mini-Programs. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.