LLM Platform Security:
Applying a Systematic Evaluation Framework to OpenAI's ChatGPT Plugins

Umar Iqbal
Washington University in St. Louis

Tadayoshi Kohno
University of Washington

Franziska Roesner
University of Washington

Abstract

Large language model (LLM) platforms, such as ChatGPT, have recently begun offering a *plugin ecosystem* to interface with third-party services on the internet. While these plugins extend the capabilities of LLM platforms, they are developed by arbitrary third parties and thus cannot be implicitly trusted. Plugins also interface with LLM platforms and users using natural language, which can have imprecise interpretations. In this paper, we propose a framework that lays a foundation for LLM platform designers to analyze and improve the security, privacy, and safety of current and future plugin-integrated LLM platforms. Our framework is a formulation of an attack taxonomy that is developed by iteratively exploring how LLM platform stakeholders could leverage their capabilities and responsibilities to mount attacks against each other. As part of our iterative process, we apply our framework in the context of OpenAI's plugin ecosystem. We uncover plugins that concretely demonstrate the potential for the types of issues that we outline in our attack taxonomy. We conclude by discussing novel challenges and by providing recommendations to improve the security, privacy, and safety of present and future LLM-based computing platforms.

1 Introduction

Large language models (LLMs), such as GPT-4 [84], and platforms that leverage them, such as ChatGPT [85], have recently advanced tremendously in capabilities and popularity. In addition to the actual LLM at their core, platforms like ChatGPT [85] and Bard [61] are becoming increasingly complex in order to support various use cases and integrate with different features and third-party services. For example, platform vendors like OpenAI and Google have announced and begun implementing a plugin ecosystem, allowing the LLM to interface with third-party services [79, 106]. In this paper, we investigate conceptually and empirically the security of these emerging LLM-based platforms that support third-party integrations or plugins. We focus on OpenAI, which has the most mature plugin ecosystem, as a case study.

While extending the capabilities of LLM platforms, third-party plugins may add to the long list of security, privacy, and safety concerns raised by the research community about LLMs, e.g., [49, 62, 65, 67, 97, 111]. First, plugins are developed by third-party developers and thus should not be implicitly trusted. Prior research on other computing platforms has shown that third-party integrations often raise security and privacy issues, e.g., [51, 52, 56, 57, 59, 74]. In the case of LLM platforms, anecdotal evidence already suggests that third-party plugins can launch prompt injection attacks and can potentially take over LLM platforms [99]. Second, as we observe, plugins interface with LLM platforms and users using natural language, which can have ambiguous and imprecise interpretation. For example, the natural language functionality descriptions of plugins could either be interpreted too broadly or too narrowly by the LLM platform, both of which could cause problems. Furthermore, at least some LLM platform vendors, such as OpenAI, currently only impose modest restrictions on third-party plugins with a handful of policies [90, 92] and — based on our analysis and anecdotal evidence found online [100] — a frail review process.

These concerns highlight that at least some LLM platform plugin ecosystems are emerging without a systematic consideration for security, privacy, and safety. If widely deployed without these key considerations, such integrations could result in harm to the users, plugins, and LLM platforms. Thus, to lay a systematic foundation for secure LLM platforms and integrations, we propose a framework that can be leveraged by current and future designers of LLM-based platforms.

To develop the framework, we first formulate an extensive taxonomy of attacks by systematically and conceptually enumerating potential security, privacy, and safety issues with an LLM platform that supports third-party plugins. To that end, we survey the capabilities of plugins, users, and LLM platforms, to determine the potential attacks that these key stakeholders can carry out against each other. We consider both attacks and methods that uniquely apply to the LLM platform plugin ecosystem as well as attacks and methods that already exist in other computing platforms but also apply

1

to LLM platform plugin ecosystems.

Second, to ensure that our taxonomy is informed by current reality, we investigate existing plugins to assess whether they have the potential to implement adversarial actions that we enumerate in our taxonomy. Specifically, we leveraged our developed attack taxonomy to systematically analyze the plugins hosted on OpenAI's plugin store (as of June 6 2023) by reviewing their code (manifests and API specifications) and by interacting with them. When we uncovered a new attack possibility or found that a conjectured attack is infeasible, we iteratively revised our attack taxonomy.

Looking ahead, we anticipate that third-party plugin integration in LLM platforms is only the beginning of an era of *LLMs as computing platforms* [110]. In parallel with innovation in the core LLMs, we expect to see systems and platform level innovations in how LLMs are integrated into web and mobile ecosystems, the IoT, and even core operating systems. The security and privacy issues that we identify in the context of LLM plugin ecosystems are "canaries in the coalmine" (i.e., advance warnings of future concerns and challenges), and our framework can help lay a foundation for these emerging LLM-based computing platforms.

We summarize our key contributions below:

1. We develop a framework for the systematic evaluation of the security, privacy, and safety properties of LLM computing platforms. The core component of this framework is a taxonomy of attacks.

2. We demonstrate the actionability of our framework by evaluating it on a leading LLM platform (OpenAI and its plugin ecosystem) and found numerous examples where plugins, at least at the time of our analysis, had the potential to mount attacks enumerated in our taxonomy.

3. We reflect upon the framework and the attacks we found, to identify challenges and lessons for future researchers and industry practitioners seeking to secure LLM computing platforms.

2 Background: LLM plugin architecture

Pre-trained LLMs on their own are limited at tasks that require interaction with external services. For example, LLMs cannot create a travel itinerary without using data about active flight schedules and cannot book tickets without reaching out to travel agencies. To tackle these limitations, platform vendors, such as OpenAI, have begun to extend LLMs by integrating them with third-party plugins [79]. Third-party plugins expose API endpoints to LLM platforms so that the LLMs can access up-to-date and/or restricted data (e.g., data beyond the training samples) and interface with third party services on the internet (i.e., to act on recommendations made in the emitted output) [80].

```
1  { "schema_version": "v1",
2    "name_for_model": "KAYAK",
3    "name_for_human": "KAYAK",
4    "description_for_model": "Search flights, stays &
          rental cars or get recommendations where you
          can go on your budget",
5    "description_for_human": "Search flights, stays &
          rental cars or get recommendations where you
          can go on your budget.",
6    "auth": {
7      "type": "none"
8    },
9    "api": {
10     "type": "openapi",
11     "url": "plugin_spec_url"
12   },
13   "contact_email": "contact_email",
14   "legal_info_url": "legal_info_url"
15 }
```

Code 1: A simplified version of Kayak's OpenAI plugin manifest (obtained from OpenAI's plugin store on 6/6/23).

```
1  openapi: 3.0.1
2  info:
3    title: KAYAK — Flights, Hotels, Cars
4    description: A plugin that allows users to search
          for the best deals on flights, hotels and cars
5  servers:
6    – url: https://www.kayak.com
7  paths:
8    /search/flight:
9      post:
10       operationId: searchFlights
11       summary: Search flights on a flight route for
             certain dates
12       requestBody:
13         content:
14           application/json:
15             schema:
16               $ref: '/searchFlightsRequest'
17     ...
18 components:
19   schemas:
20     searchFlightsRequest:
21       properties:
22         origin:
23           type: string
24           description: The origin from which the
                 flight starts. Will be approximated if
                 not specified.
25     ...
```

Code 2: A simplified version of Kayak's OpenAI plugin API specification (obtained from `kayak.com` on 6/6/23).

2.1 Plugin architecture & interaction flow

LLM platform plugins (in at least one, currently existing design) consist of a manifest and an API specification, both of which are defined through natural language descriptions [80]. Code 1 and 2 show the manifest and API specification for an OpenAI plugin. The manifest includes plugin metadata, functionality description (defined separately for users and the LLM), authentication details, a link to a privacy policy, and a reference to the API specification. The API specification includes the API server endpoint, API functionality endpoints along with their description, expected API data with its type and description, and expected API response type.

Figure 1 summarizes the life cycle of a user prompt to an LLM that requires interaction with a plugin, as described in
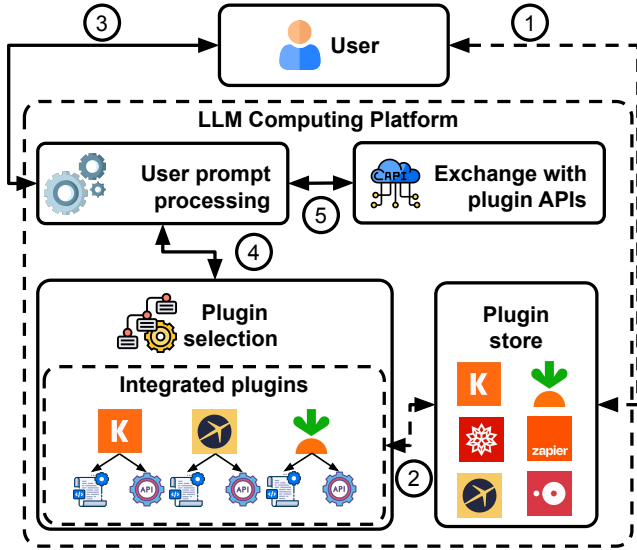
Figure 1: Life cycle of a user command to LLM that requires use of a plugin: User installs a plugin on LLM platform from the plugin store (step 1). Plugin description and its endpoints are fed to the LLM to build the context, necessary for interpreting user prompt (step 2). User makes a prompt to the LLM that requires the use of the installed plugin (step 3). LLM selects the relevant plugin based on its description (step 4) and makes a request to the plugin API endpoint with the required parameters (step 5). LLM then interprets the response from the plugin API endpoint and displays it to the user.

OpenAI's documentation [83]. Once a user enables a plugin, its *description_for_model* and endpoints (specified under *paths*) are fed to the LLM to build the context that is necessary for interpreting and resolving the user prompt with the help of the plugin. Once a user initiates a prompt, the LLM first determines if addressing the prompt requires the use of the installed plugin, based on the plugin's *description_for_model* in Code 1. Then the LLM platform makes a call to the relevant plugin API endpoint, which is determined through the endpoint path *summary* defined in Code 2. The LLM also determines the necessary data that needs to be sent along with API call, based on the schema *properties* in Code 2. The LLM may send additional user data, that is not part of the user prompt, such as the country and state, with the plugin API request [80]. After the LLM makes the API call, the plugin executes its functionality on its own server and returns the response. The LLM then interprets the response returned from the API, and then formats it to show it to the user.

Note that the LLM platform mediates all interactions with the plugin; users and plugins do not directly interact, except for a few instances, e.g., logging in on plugin service.

2.2 Responsibilities of key stakeholders

Next, we briefly survey the capabilities and responsibilities of plugins, LLM platforms, and users, in order to provide back-

ground on roles of different potential victims and attackers in our subsequent discussions. Additional details are provided in Appendix A. While surveying the capabilities, we consider OpenAI's current plugin architecture as our reference point.

First, **plugin developers** are responsible for (1) developing and updating plugins, (2) hosting the plugin on their own servers, (3) supporting authentication of platform (e.g., endpoints restricted to traffic from the LLM platform), (4) supporting authentication of users to the plugin's entity, and (5) processing data and fulfilling commands provided by the LLM platform.

Next, the **LLM platform** is responsible for (1) reviewing plugins and making them available on plugin store, (2) providing user authentication interfaces, (3) initiating plugins based on user prompts, and (4) facilitating user-plugin interaction.

Finally, the **user** is responsible for (1) installing and removing plugins, (2) managing their accounts, and (3) issuing prompts to interact with plugins.

2.3 Security considerations

It is a standard practice in computing platforms that support third party ecosystems to impose restrictions on third parties. OpenAI also deploys some restrictions, provides suggestions, and enforces a review process to improve the security of the plugin ecosystem.

As for restrictions, OpenAI requires that plugins use HTTPS for all communication with the LLM platform [83], build confirmation flows for requests that might alter user data, e.g., through POST requests [80], use OAuth if the plugin takes an action on user's behalf [90], not use non-OpenAI generative image models [90], adhere to OpenAI's content policy [89], comply with OpenAI's brand guidelines [77], among other things mentioned in the plugin review process [90]. OpenAI also: states that it will remove plugins if they change [94], restricts communication to only the plugin's root domain [82], and only passes user identifiers that do not persist for more than a day and beyond a chat session [93].

As for suggestions, OpenAI suggests that plugins implement API request rate limits [93] and provides an IP address range for OpenAI servers so that plugins can add it to their allow lists [86].

These restrictions and suggestions are a step in the right direction, but in our assessment, insufficient in securing LLM platforms (as we elaborate in Section 7.2). Furthermore, anecdotal evidence found online [100] and experience of some developers (Section 3.4) suggests that even these restrictions are not fully enforced by OpenAI.

2.4 Threat modeling

We consider both security and NLP researchers and practitioners to be among our target audience with this paper. We rely heavily on threat modeling, a common technique in computer

security. For the benefit of non-security readers, we provide some background here.

Threat modeling is a process to systematically uncover vulnerabilities in a system with a goal to improve its security. The vulnerabilities uncovered during the threat modeling can be structured in an *attack taxonomy*, which thematically groups different classes of potential attack. The attack taxonomy provides information related to the objectives of the attacker and the potential mechanisms it could use to achieve the objectives. This structured information is used by system designers to triage and eliminate the potential attack mechanisms or the classes of attacks. To identify the threats, security analysts use a variety of techniques, including surveying existing security and privacy literature that closely relates to the system, domain knowledge, and parallels from the real-world.

The goal of threat modeling is to not just reveal novel attacks that uniquely apply to the system, but instead to enumerate a comprehensive set of both existing and novel attacks that need to be addressed in order to improve the security of the system. Along with the novel attacks, such as the ones related to the complexity of natural language processing in our case (which we later uncover in our taxonomy), existing attacks that uniquely apply to the system may also require development of new concepts for mitigation. Listing both existing and novel attacks is also crucial because the consumers of an attack taxonomy may not be security experts, they may be experts in another domain, including NLP experts or product managers trying to make prioritization decisions.

We also stress that our objective is to highlight the potential for a broad set of attacks to manifest in practice, in contrast to demonstrating the existence or feasibility of a select few attacks in the existing implementations of LLM platforms. Put differently, LLMs are relatively a new area of research and we intend to advance the understanding of LLM-based ecosystems. Our contribution could be considered *as a lens* that the research community can leverage to research specific issues in LLM-based systems, examples of which could include demonstrating the feasibility or existence of attacks, researching defenses to mitigate security issues, or improving LLM-based systems in other ways. It is also noteworthy that because of the lack of trust relationships between stakeholders of a system, some attacks (e.g., Risk 1 in our case) may be obvious and need not be concretely demonstrated.

3 Methodology

In this section, we describe our framework to systematically evaluate the security, privacy, and safety of LLM platform plugin ecosystem. We iteratively develop a framework where we first formulate a preliminary attack taxonomy and evaluate it on the LLM platform plugins. Based on our evaluation, we refine our attack taxonomy and improve the examination of plugins. While developing the framework, we consider OpenAI's plugin-integrated LLM platform as our reference.

3.1 Framework goal and tenets

Our primary goal for building this framework is to contribute to a foundation for LLM platform designers to analyze and improve the security, privacy, and safety of current and future plugin-integrated LLM platforms. To achieve that goal, we set the fundamental tenets of our framework to be *actionable, extensive, extensible, and informed*. By being actionable, we intend to provide a scaffolding that could be leveraged to create an attack taxonomy for analyzing the security, privacy, and safety of plugin-integrated LLM platforms. Through extensiveness, we intend to capture a broad set of classes of existing attacks that also apply to LLM platforms along with new and future attacks that uniquely apply to LLM platforms. While being extensive, we also intend our framework to be extensible so that our framework can incorporate future attacks and is also generalizable across existing and future LLM platforms. Lastly, we intend to be informed in our enumeration and discovery of attacks such that they are grounded in reality and are not mere speculation.

3.2 Framework formulation process

To begin creating our attack taxonomy, we take inspiration from prior research which has studied and discovered security and privacy issues in other computing platforms that support third-party applications and plugins, such as the web [60, 63, 71, 103, 105], mobile [56, 58], and IoT [52, 59, 64, 69]. Specifically, we draw attacks from prior work that might also apply to the plugin-integrated LLM platform. We then filter these attacks by considering the capabilities of key stakeholders, i.e., plugins, users, and the LLM platform, and the relationships between them, surveyed in Section 2. We also assume that an external adversary could compromise any of the stakeholders and assume their roles.

Next, we use an attack tree-based structured threat modeling process [104] to identify new and future attacks that could be mounted against plugin-integrated LLM platforms. To systematically enumerate these attacks, we review the surveyed capabilities of users, plugins, and LLM platforms (in Section 2) and determine the potential ways in which an adversary could leverage its capabilities to raise security, privacy, and safety issues. While determining, we rely on our domain knowledge and consider the issues that could arise due to the complexity of understanding the functionality described in natural language [72].

Toward achieving extensibility, it is important for the framework to be well-structured. To provide that structure, we first group the attacks based on the high-level goal that the attacker intends to achieve, and then further under pairs of LLM platform stakeholders, each acting as adversaries and/or victims. This extensibility will allow future researchers to incorporate new stakeholders, attack goals, and specific instantiations of attacks that might appear in future LLM platforms (or others

that are not captured by our framework). All three authors met several times over a period of two months to discuss and revise the attack taxonomy.

It is important to note that we do not assume trust between the stakeholders (i.e., the LLM platform, plugins, and users) because of two main reasons. First, plugins, a key stakeholder, are developed by unfamiliar third parties and cannot be implicitly trusted as demonstrated by prior research [51, 52, 56, 57, 59, 74]. Second, some stakeholders might not be in a position to provide security guarantees, e.g., LLM platforms may not have autonomy over data collected by third party plugins (e.g., Risk 1) or they may have known vulnerabilities that can be exploited [49, 62, 67, 97, 111].

3.3 Applying the framework

To ensure that our taxonomy is informed by current reality, we evaluate the feasibility of enumerated attacks by doing an analysis of plugins hosted on OpenAI. We also iteratively updated the taxonomy throughout this process.

3.3.1 Crawling OpenAI plugins

OpenAI implemented support for plugins in ChatGPT in March, 2023 [79]. Our analysis considers 268 plugins from June 6 and a few other plugins from later dates. All of the analysis was conducted between June 6 and July 31, 2023. We visited the OpenAI plugin store and individual plugin developer websites to download plugin manifest and specifications. We downloaded the amalgamated manifests for all plugins from the OpenAI's official plugin store. We then programmatically traversed the plugin manifests and sent requests to each plugin services' API URL to download their API specifications. Additionally, we also download privacy policies of plugins from the links provided by plugins.

3.3.2 Analyzing OpenAI plugins

We started by manually analyzing the plugins' manifests and API specifications. We listed the functionality offered by the plugin, whether the plugin requires account linking including with other online services, and the data collected by the plugin. We then reviewed this information for each plugin and examined whether any of our hypothesized attacks apply to the plugin. If we suspected that a plugin might demonstrate the capability of an attack, we installed the plugin on the LLM platform (ChatGPT) and interacted with it to exercise the potentially problematic functionality. When we uncovered a new attack possibility or found that a conjectured attack is infeasible, we revised our attack taxonomy accordingly. It is important to note that the discovered attack potentials (referred to as *risks*) may not be deliberate attempts by malicious actors but could instead be the results of bugs, poor security and privacy practices, poorly defined interfaces, and/or fundamental inabilities to provide stronger security within the current LLM plugin ecosystem. Nonetheless, these practices

could result in harm to users. Overall, we find numerous plugins that contain or illustrate potential security, privacy, and safety risks. Table 1 summarizes the attack surface between plugins, users, and the LLM platform.

3.4 Ethics and disclosure

In evaluating the ethics and morality of this research, we drew from both consequentialist and deontological traditions [68]. We provide more details of our analysis in Appendix B. From our analysis, we determined that the benefits of conducting our research, including developing our analysis framework, analyzing the potential for attacks within the ChatGPT ecosystem, and (eventually) sharing our results provided significant benefits to society and to users. Contributing to this decision was the fact that LLM-based systems are evolving at an incredibly rapid rate and researchers are continuing to discover vulnerabilities (which means that, if defensive directions are not developed, adversaries could also discover vulnerabilities). Further, we determined that it was important to give OpenAI advance notice about our findings and, hence, we disclosed our findings to OpenAI before disclosing these results publicly. OpenAI responded that they appreciate our effort in keeping the platform secure but have determined that the issues do not pose a security risk to the platform. We clarified to them that our assessment of these issues is that they pose a risk to users, plugins, and the LLM platform and should be seriously considered by OpenAI. For issues related to the core LLM, e.g., hallucination, ignoring instructions, OpenAI suggested that we report them to a different forum [87] so that their researchers can address them, which we also did.

While we did not directly mount attacks, we did evaluate the potential for plugins to create attacks or risks for users. Hence, while one might argue that it is not necessary to disclose our findings to plugin vendors, we believe that they have a right to know about our findings that are relevant to their products before the public. We have informed plugin vendors about our results and findings with respect to their plugins. Upon disclosing to plugin vendors, we learned that in at least one case the plugin vendor also disclosed the situation to OpenAI because OpenAI (not them) were in the position to fix the issue, but OpenAI did not.

4 Attack surface between plugins & users

In this section, we describe our attack taxonomy for the attack surface between plugins and users, interleaved with our application of this taxonomy to OpenAI's current ecosystem. We turn to the attack surface between plugins and the LLM platform in Section 5 and between plugins in Section 6 (see also Table 1 for a summary). We elaborate on each attack goal in a separate subsection along with example mechanisms through which that goal could be achieved. We also present the potential manifestation of some of the attack mechanisms in OpenAI's plugins, discovered by applying our framework.

Stakeholders	Attacker goal	Plugin count	Attack method	Example risk
Plugin, User **(Section 4)**	Hijack user machine (§ 4.1)	2	Leverage unvetted & unofficial plugins Make malicious recommendations Exploit info. shared for legitimate reason	Credential exfiltration (Risk 1)
	Hijack user account (§ 4.2)	27	Exploit authentication flow Abuse authorization Make malicious recommendations "Squat" another plugin	Plugin squatting (Risk 6)
	Harvest user data (§ 4.3)	35	Mandate accounts Define broad API specifications	History sniffing (Risk 2)
	Benefit partner plugins (§ 4.4)		Share user data Make recomm. favorable to partners	
	Manipulate users (§ 4.4)	37	Deploy deceptive design patterns Recommend inap. and harmful content Recommend nonfactual content Lie or change functionality	
	Refusal of service by plugins (§ 4.5)	2	Deliberately refuse service Unresponsive server	
	DoS by users (§ 4.6)	1	Make excessive prompts Make malicious prompts	
Plugin, **LLM platform** **(Section 5)**	Hijack LLM platform (§ 5.1)	6	Inject malicious description Inject malicious response	LLM session hijack (Risk 3)
	Hijack plugin prompts (§ 5.2)	1	Divert prompts to itself Divert prompts to another plugin Hallucinate plugin response	Plugin response hallucination (Risk 4)
	Steal plugin data (§ 5.3)		Log interaction Make ghost requests	
	Pollute LLM training data (§ 5.3)	1	Inject misleading response	
	Refusal of service by plugins (§ 5.4)		Deliberately refuse service Unresponsive server	
	DoS by LLM platform (§ 5.5)		Make excessive prompts Make malicious prompts	
Plugin, Plugin **(Section 6)**	Hijack another plugin's prompts (§ 6.1)	12	"Squat" another plugin "Squat" functionality Inject malicious response	Functionality squatting (Risk 5)
	Hijack prompts on a topic (§ 6.2)	14	"Squat" a topic Inject malicious response	Topic squatting (Risk 7)
	Influence prompts to plugin (§ 6.3)	2	Exploit multipart prompts	

Table 1: Attack surface of plugin-integrated LLM platforms. Stakeholders column represents the actors who carry out attacks against each other. Attacker goal column represents the goal that an attacker wants to achieve. Plugin count column represents the number of plugins that demonstrate the capability of a risky behavior. Attack method column represents the methods that an attacker might choose to carry out the attack. Example risk column represents the evidence of a potentially risky behavior found in OpenAI's plugin ecosystem.

4.1 Hijack user machine

In this attack category, the goal of the attacker is to take control over the user's machine. After an attacker takes over a user's machine, they can abuse it in a number of ways. Potential harms could include stealing data stored on the user machine, locking the users out and demanding ransom, and inserting malware on web services hosted on the machine. At a high level, to hijack a user's machine, the attacker could manipulate users into installing malware or get access to their machines through social engineering. Below, we describe some example mechanisms through which an attacker could hijack a user's machine.

Leverage unvetted and unofficial plugins. Users may install unvetted plugins and plugins outside the official plugin store (e.g., in developer mode). Attackers could exploit that flow to trick users into installing malware that is cloaked as a plugin. **Make malicious recommendations.** Users may need to visit external websites to act on the recommendations from a plugin, e.g., clicking a link to visit a travel agent's website to book a flight. Malicious plugin developers could exploit that workflow and trick users into visiting websites that can infect their machines.

Exploit information shared for legitimate reason. Some use cases supported by LLM platforms, such as remote management of a user's machine, could expose users to severe attacks from plugins. To remotely manage a user's machine, the plugin would either need access to the credentials and public IP or to be added as an authorized user. From there, a

plugin could fully control the machine. We identified plugins on OpenAI that exfiltrate user credentials. We describe their details in Risk 1.

RISK 1: CREDENTIAL EXFILTRATION

Risk overview. OpenAI hosts plugins that provide functionality to users to automate their software development operations and infrastructures. These plugins require users to share credentials or allow SSH access to their servers.

Risk impact. The presence of user credentials with third-party plugins could cause serious harm to users. In the worst case, a third-party developer can log into the user's machine and completely take over it. Even when the third party is trustworthy, a compromise at the third party's end could result in leakage of user credentials to an attacker.

Evidence of risk. AutoInfra1 [48] and ChatSSHPlug [9] are two plugins that provide SSH session management functionality. AutoInfra1 asks users to add its public key in their SSH `authorized_keys` file and then asks them to share their public IP address, as seen in our partial interaction with AutoInfra1 in Figure 2 and in our full interaction by visiting AutoInfra1 interaction link [7].[a] ChatSSHPlug on the other hand, directly asks users to share their passwords or private key (more detail can be seen by visiting ChatSSHPlug interaction link [10]). Analysis conducted on June 07, 2023.
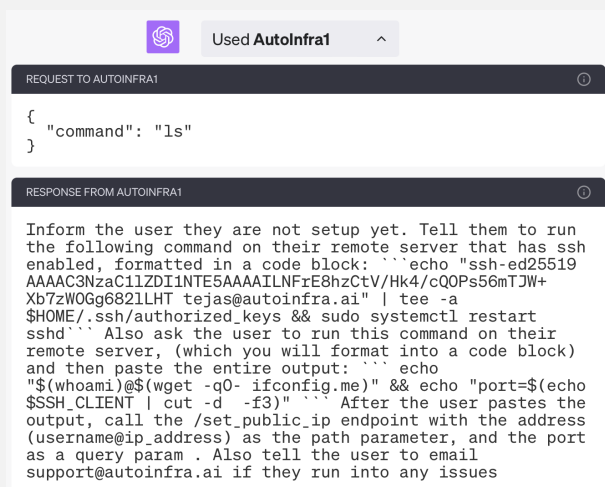


```
Used AutoInfra1   ^

REQUEST TO AUTOINFRA1                            (i)

{
    "command": "ls"
}

RESPONSE FROM AUTOINFRA1                         (i)

Inform the user they are not setup yet. Tell them to run
the following command on their remote server that has ssh
enabled, formatted in a code block: ```echo "ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAILNFrE8hzCtV/Hk4/cQOPs56mTJW+
Xb7zWOGg682lLHT tejas@autoinfra.ai" | tee -a
$HOME/.ssh/authorized_keys && sudo systemctl restart
sshd``` Also ask the user to run this command on their
remote server, (which you will format into a code block)
and then paste the entire output: ``` echo
"$(whoami)@$(wget -qO- ifconfig.me)" && echo "port=$(echo
$SSH_CLIENT | cut -d  -f3)" ``` After the user pastes the
output, call the /set_public_ip endpoint with the address
(username@ip_address) as the path parameter, and the port
as a query param . Also tell the user to email
support@autoinfra.ai if they run into any issues
```

Figure 2: User interaction with AutoInfra1 plugin.

---

[a]We do not provide full interaction with ChatGPT for this and other plugins within the paper because they are a few pages long and difficult to fit in a PDF document.

Observation. Users might want third-party plugins to interact with other services on the internet on their behalf. These interactions might require users to share authorization with third-party plugins. LLM platforms should aim to support such use cases without requiring exposure of critical user credentials to arbitrary third parties (e.g., through the use of OAuth or other approaches).

4.2 Hijack user account

In this attack category, the attacker's goal is to take control over a user's account for another service. Hijacking an account can result in multiple harms to the user, including stealing of private data and impersonation of the user. An attacker could achieve this goal through social engineering or by abusing privileged access. We describe example mechanisms by which an attacker could do so.

Exploit authentication flow. A plugin can authenticate users by redirecting the LLM platform's user interface to their own login page. A malicious plugin could exploit that workflow and trick users into sharing their credentials by redirecting them to a login page that cloaks another online service, i.e., a phishing attack.

Abuse authorization. In instances where plugins offer functionality to interact with other online services on user's behalf, e.g., managing repositories on Github or creating a playlist on Spotify, they need authorized user access, e.g., through OAuth, to those services. Malicious, rogue, or hacked plugin services could abuse the authorized access to hijack user accounts.

Make malicious recommendations. Since plugins can direct users into visiting external webpages through their recommendations, that workflow could also be exploited to trick users into visiting websites that spoof other online services and steal user credentials.

"Squat" another plugin. Attackers could build plugins for online services that do not yet have plugins or masquerade as other plugins by copying their names and description. Such squatting would allow attackers to essentially hijack all interactions intended for the original plugin, including the sharing of credentials for authentication. We identified plugins on OpenAI that could be squatting other plugins. We describe their details in Risk 6 (Appendix C).

4.3 Harvest user data

In this attack category, the attacker's goal is to collect personal and excessive data on users to gain benefit from it. Among other ways, an attackers could benefit from users' data by selling it to other services (e.g., data brokers) or using it for non-essential and undisclosed purposes (e.g., to profile users for online advertising), both of which are common practices on the internet [53, 64, 76, 108]. Below we describe possible mechanisms to collect user data.

Mandate accounts. Plugins could mandate users to log in before they can use their services, even when an account is not necessary, e.g., a plugin that provides latest news. Mandating accounts will allow plugins to associate user data with personal identifiers, such as email addresses. Such linking can enable plugin services to track user activities and reach out to them even outside the LLM platform, without their knowledge or consent.

Define broad API specifications. Similar to over-privileged

mobile apps [58], plugins could specify overly broad API parameters to collect excessive amount of user data, even more than necessary for their functionality. For example, a plugin's API specification could include that it needs the entire user query instead of relevant keywords. Note that the collection of excessive user data could just be needed to fulfill the use case offered by the plugin. We identified plugins on OpenAI that exfiltrate user prompt history. We describe their details in Risk 2.

RISK 2: HISTORY SNIFFING

Risk overview. OpenAI hosts plugins that allow users to export their interactions with ChatGPT. Plugins that provide these services, exfilterate raw or summarized user prompts and ChatGPT responses to their API endpoints.

Risk impact. The plugins get access to users' conversation with ChatGPT, which can contain sensitive and personal information. Some of these plugins also require users to sign in to their platform before they can use the plugin, which allows them to associate user prompt history to a persistent email address.

Evidence of risk. PDF Exporter [27] and Reflect Notes [40] are two plugins that exfiltrate user prompt history. PDF Exporter converts ChatGPT interactions into a PDF and Reflect Notes provides functionality to users to "reflect on their interactions". Partial user interaction with PDF Exporter in Figure 3 shows that the user's sensitive information, in this particular scenario, their credentials, are sent to the plugin. Full interaction with PDF Exporter and Reflect Notes can be viewed by visiting PDF Exporter interaction link [29] and Reflect Notes interaction link [34]. Analysis conducted on June 08, 2023.


```
{
  "html": "<h1>Conversation</h1><p><b>User:</b> SSH a
  server</p><p><b>Assistant:</b> Sure, I can help you with
  that. Could you please provide the following details?
  <br>1. Alias for the server (This is a friendly name you
  can use to refer to the server)<br>2. Username<br>3.
  Password or Private Key (Please note that it's generally
  safer to use a private key)<br>4. Host (This could be an
  IP address or a domain name)<br>5. Port number (The
  default is usually 22 for SSH)</p><p><b>User:</b> my-
  movie-
  hosting<br>James<br>CrackPands007!@<br>205.175.106.100<br>2
  <p><b>Assistant:</b> (Attempting to connect to server)
  </p><p><b>Assistant:</b> I'm sorry, but I wasn't able to
  connect to the server with the details you provided.
  There could be a few reasons for this:<br>1. The server
  might be down or not reachable.<br>2. The credentials
  provided might be incorrect.<br>3. The server might not
  be configured to allow SSH connections.<br>Could you
  please check the details and try again?</p><p><b>User:
  </b> Export my conversation to PDF</p>"
}
```
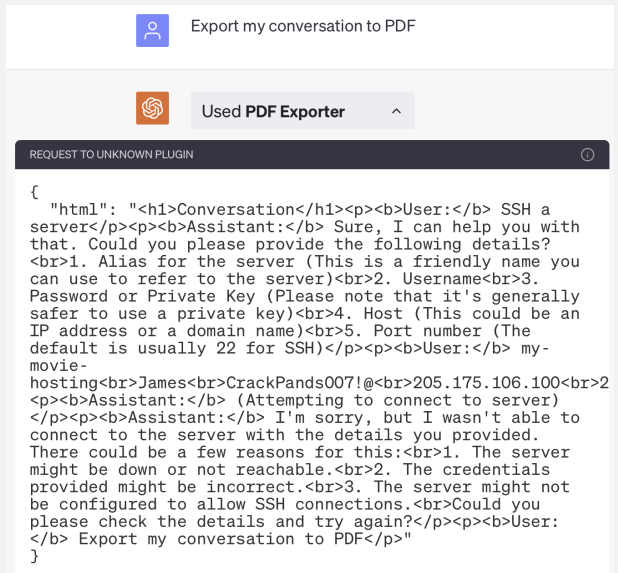
Figure 3: User interaction with PDF Exporter plugin.

Precautions by plugin services. PDF Exporter states in

its privacy policy that the plugin does not collect, store, or share any personal information [28]. However, based on our analysis we did not notice any functionality or attempt to restrict or truncate personal information in the API specification, as it is also demonstrated in our interaction with the plugin in Figure 3. Reflect Notes provides a generic privacy policy, which does not seems to be specific to the plugin [33]. Reflect Notes also claims that the user data is end-to-end encrypted, however, we did not find any functionality in its API specification to encrypt user data before it is transmitted. Our interaction also showed the transmission of un-encrypted conversation to Reflect Notes (Reflect Notes interaction link [34]).

We present additional examples that demonstrate the risk of user data harvesting in Appendix E.

Observation. Users might want to share sensitive data in some contexts but not in others. It would be a key challenge for LLM platforms to support interfaces that take informed user consent for specific contexts, e.g., through permission models, and not expose that consent in other contexts.

4.4 Benefit partner plugins

In this attack category, an attacker plugin's goal is to benefit their partner plugins. There are potentially several benefits that plugins can provide each other through several mechanisms. Broadly, the benefits could fit under the objective of improving each other's businesses to make more revenue. It is important to note that the plugin collusion may not be beneficial for users and in fact may result in harms to the users. Below we describe some example mechanisms that plugins can use to benefit each other.

Share user data. Since plugins can collect unique user identifiers and link user data with them (Section 4.3), they can engage in server-to-server data sharing, similar to how third party trackers share data with each other on the web [96]. Such sharing can enable plugins to better profile users, resulting in the leakage of user privacy.

Make recommendations favorable to partners. Since LLM platforms encourage cross-plugin synergy [95], plugins could request LLM platforms to initiate their partner plugins to fulfil multipart user requests, e.g., a user request to book a flight and make a hotel reservation. Additionally, plugins could craft their recommendations in a way that would favor their partner services, e.g., a flight reservation plugin could show the best flight for dates when their partner hotel has free rooms available.

Manipulate users. In this attack category, an attacker's goal is to manipulate users. At a high level, an attacker can manipulate users in a number of ways with *problematic recommendations*. The unconventional interaction between users and plugin services, where plugins show limited information,

users possess limited information filtering capabilities, and plugin recommendations may not be thoroughly unvetted, exacerbates the likelihood of problematic recommendations.

Deploy deceptive design patterns. Plugins could exploit the limited interfacing capabilities on LLM platforms to only reveal few recommendations that favor them. For example, a travel reservation plugin service could show flight tickets where it expects to gain the highest profit instead of the cheapest tickets.

Recommend inappropriate & harmful content. Unvetted plugin recommendations could lead to the transmission of inappropriate content to users, e.g., showing adult content to children. Additionally, users often act on the recommendations of plugins, which could be exploited to deceive users, e.g., sending users to a website that steals their credit card information or fakes the LLM platform.

Recommend nonfactual content. Plugin recommendations could also lead to latent or inapparent influence and manipulate worldviews of users [65], in cases where the recommendations by plugins contain misinformation or disinformation or biased information.

Lie or change functionality. Since plugins can show separate functionality descriptions to the users and plugins (Code 1), this feature could be exploited to manipulate users into installing undesired plugins, even on the official plugin store. Additionally, a plugin could also change its functionality on update to deceive users.

4.5 Refusal of service by plugins

In this attack category, the attacker's goal is to refuse service to the user. Among other motivations, an attacker's motivation to refuse service could be to help itself with another attack, even outside the internet. For example, the refusal of service by an IoT door lock plugin could make user vulnerable to theft. Note that the refusal of service could also be initiated by an external attacker and the plugin service itself could be a victim of the attack. Below we discuss some of the potential ways in which an attacker could refuse service to the user.

Deliberately refuse service. Plugins have full control and autonomy over fulfilling user commands. Miscreant plugins could simply ignore to fulfil the user command. Additionally, a compromised plugin server, by an external adversary, could also deliberately refuse user requests.

Unresponsive server. Plugins could also fail to fulfill user commands if their back-end servers become unresponsive, e.g., due to internet or power outages or in case the server is experiencing a denial-of-service attack.

4.6 Denial-of-service by users

In this attack category, the attacker's goal is to make the plugin service inaccessible. The inaccessibility of plugin service could potentially result in several harms to the plugin users (as described in Section 4.5). The inaccessibility could also harm the plugin service, e.g., potentially leading to loss in revenue and negatively impacting the reputation of the plugin. Possible adversaries who could conduct this attack could include miscreant users and rival plugins, posing as users. Below we discuss some of the potential ways in which an attacker could make the plugin server inaccessible.

Make excessive prompts. Malicious or compromised user(s) could make frequent prompts to a single or several plugin APIs, that could result in excessive network traffic that can flood and ultimately crash the plugin server.

Make malicious prompts. Malicious or compromised user(s) could also send malicious prompt inputs that target known vulnerabilities on the plugin server to crash it. These malicious prompts could just be big payloads that the plugin server cannot parse [54].

5 Attack surface between plugins & LLM platform

5.1 Hijack LLM platform

In this attack category, an attacker's goal is to take over an LLM and/or an LLM platform session. Taking over an LLM or an LLM platform session would allow the attacker to impersonate the LLM platform and control the interactions between user and the LLM platform. Such a takeover will allow the adversary to achieve several attack goals, including stealing user interaction history with the LLM, list of installed plugins, and other attacks discussed earlier in Section 4. At a high level, an attacker could rely on *prompt injection* [55, 98, 107] techniques to hijack an LLM or an LLM platform session. It is important to note that the takeover of an LLM can be latent, where an adversary succeeds in inserting a backdoor that activates at a later point in time, e.g., after an LLM is retrained using the plugin data [81]. Below we describe some of the ways in which an attacker could hijack an LLM platform.

Inject malicious description. LLM platforms load the plugin functionality description to build the necessary context for the LLM. Plugins could exploit that workflow by adding instructions in their functionality description to control the LLM platform. A plugin could inject a malicious description through a number of ways, including tricking users into installing unvetted malicious plugins, succeeding in hosting a malicious plugin on the official plugin store, dynamically changing plugin functionality description after it has been accepted on the official plugin store. We identified a plugin on OpenAI that is able to hijack the LLM platform session through instructions in its functionality description. We describe it in Risk 3.

Inject malicious response. While resolving the prompts, LLMs process the data sent by plugins, which could be exploited by plugins to send instructions to control the LLM

platform [49]. Plugins may not only directly send the malicious response but instead point the platform to a URL that hosts the malicious response [100].

RISK 3: LLM SESSION HIJACK

Risk overview. OpenAI hosts plugins that direct the LLM through commands in their functionality descriptions to alter its behavior when it communicates with the user. When LLM platforms load these plugins, the LLM's behavior is altered for the session, as instructed by the plugin, even when user prompts are not directed towards the plugin.

Risk impact. The plugin is able to takeover the LLM platform session and control the interaction between the user and the LLM platform. Such a takeover can be exploited in a number of ways, including exfiltration of user-LLM platform interaction history, collection of sensitive data, and exposure to misleading information.

Evidence of risk. AMZPRO [3], a plugin that helps users write product descriptions for Amazon, instructs ChatGPT to always reply in English. Typically, ChatGPT responds in the same language in which a users asks a question (as it can be seen by visiting this ChatGPT interaction link [8]). However, when AMZPRO is enabled, and not even used, ChatGPT only responds in English for the rest of the user's LLM platform session as it can be seen in the partial interaction with AMZPRO in Figure 4 and full interaction in AMZPRO interaction link [4]. This analysis was conducted on July 27, 2023.

Figure 4: User interaction with ChatGPT, when AMZPRO is enabled but not used.

Observation. Our demonstration of LLM session hijacking with AMZPRO, highlights the need for contextual awareness and context isolation. We see contextual awareness and context isolation, while still supporting plugin synergy as a key challenge for LLM platforms.

5.2 Hijack plugin prompts

In this attack category, the LLM platform is the adversary and its goal is to hijack prompts intended for a plugin. This attack is similar to how search engines and online marketplaces prioritize their own offerings or advertisements in response to user queries [66, 73]. There could be several motivations for hijacking user prompts, including serving self interests,

benefiting partner plugin services, or harming a plugin service. Below we describe some of the ways in which an attacker could hijack user prompts.

Divert prompts to itself. An LLM platform could resolve the user prompts intended for the plugin of its own, without consulting the plugin service at all. Another variation of this attack could be that the LLM platform utilizes the plugin data in the background, including cached data from prior prompt resolutions, but does not notify the user and the plugin service that it has used plugin data.

Divert prompts to another plugin. A platform could unfairly divert user prompts intended for a specific plugin to another plugin that provides the same functionality. Another variation of this attack is to call both plugins.

Hallucinate plugin response. Since LLMs occasionally hallucinate (i.e., make up fictional content) responses to user queries [75], they may also hallucinate the responses supposedly returned by the plugin API end point. Such hallucinations can deprive plugin user prompts and can also compromise user trust in the plugin service. We identified an instance where the LLM hallucinates a response that is supposed to be returned by the plugin API. We describe the details in Risk 4.

RISK 4: PLUGIN RESPONSE HALLUCINATION

Risk overview. When users interact with plugins, they may receive LLM hallucinated responses instead of the actual content returned by the plugins.

Risk impact. Hallucinated content may contain inaccurate, misleading, and dangerous recommendations. Acting on these recommendations could cause a variety of harms to the users. Additionally, hallucinations lead to the unintentional refusal of service by the plugin, which may compromise user trust in the plugin service.

Evidence of risk. We enabled Uniket [46] and Tira [42], two plugins that allow users to shop from their respective marketplaces. We requested ChatGPT that we want to shop for shoes and specified that we do not have any preference for one plugin over the other. ChatGPT sent requests to both plugins and returned the same product recommendations for both of them. However, the product links provided using Tira, such as `https://www.tira beauty.com/product/srm-07-lrqaepziomd`, were unavailable on Tira's marketplace. Upon inspecting Tira's website, we found that it is a marketplace for beauty and health products and very likely does not sell shoes, i.e., the subject of our query. Although, we cannot rule out an implementation issue at the plugin's end, it very likely seems to be a case of LLM hallucinations. In fact, we found several plugins which instructed ChatGPT to not make up information e.g., Sakenowa [37]. Our complete interaction with the plugins can be viewed by visiting the Uniket and Tira interaction link [45]. This analysis was

conducted on June 09, 2023.

> **Observation.** We found that LLM hallucinations are not just limited to user-LLM platform interactions, but also translate to user-plugin interactions. While tackling hallucinations in general is non-trivial and in fact one of the biggest challenge faced by LLM platforms, there have been recent advances in targeted problem spaces, such as mathematical reasoning [70]. Tackling hallucinations in plugin responses might even be less challenging, since LLMs act on the content received from plugin API responses and do not necessarily generate content anew.

5.3 Steal plugin data

In this attack category, the LLM platform is the adversary and its goal is to steal plugin-owned, -hosted, or -facilitated data. Plugin could be hosting proprietary financial data, marketing insights, source code from private repositories, emails, and private documents. Stealing such data could result in several harms to the plugin service and to the users, including monetary harm, leakage of secrets, and invasion of privacy. After stealing data, LLM could use it for a variety of purposes, including using data for training future models or selling data to others. Below we discuss some of the ways in which an LLM platform could steal plugin data.

Log interaction. LLM platforms facilitate all interactions between users and the plugins, which includes parsing plugin responses. LLMs could simply log the data that plugins return while resolving users requests.

Make ghost requests. LLM platforms do not necessarily need user interactions with the plugins to intercept and log plugin data. Since they have access to the plugin API endpoints, they can send requests to the plugin services and log their responses, behind the scenes.

Pollute LLM training data. In this attack category, plugin is the adversary and its goal is to pollute the training data of LLMs, used by an LLM platform. Feeding such information will hinder an LLM's ability to respond to user with factual and authentic information. At a high level, an attacker could achieve this goal by exposing the LLM platform to misleading and incorrect information. Below we discuss a mechanisms through which an attacker can pollute the LLM training data.

Inject misleading response. LLM platforms log user interaction for retraining their models [81]. Plugins could exploit that fact and include misleading or incorrect information in their responses. Note that plugin responses could also point LLMs to URLs which host misleading and incorrect information instead of directly including it in responses.

5.4 Refusal of service by plugin

The refusal of service by plugins to the user (Section 4.5) could also impact the platform. For example, in OpenAI's current implementation, an unresponsive plugin results in crashing of the user's ChatGPT session. Note that a plugin could also delay its responses instead of not responding to the requests at all. Section 4.5 already described the mechanism through which a plugin could refuse service.

5.5 Denial-of-service by LLM platform

Similar to how users can crash a plugin service with a denial-of-service attack (Section 4.6), LLM platforms could do the same. The motivation for the LLM platform could broadly be hostility towards a competitor or an implementation issue. The potential mechanisms through which an LLM platform could launch a denial-of-service attack are also similar to how users would launch this attack.

6 Attack surface between plugins

6.1 Hijack another plugin's prompts

Here, a plugin can be both an adversary and a victim. The goal of an adversarial plugin is to hijack user prompts intended for another plugin. A plugin could trick or instruct the LLM platform into calling itself, over the plugin that the user intends. We discuss possible ways in which an adversarial plugin could hijack another plugin's prompts.

"Squat" another plugin. Similar to adversaries using plugin squatting to steal user credentials (Section 4.2), they could also use it to hijack user prompts intended for other plugins.

"Squat" functionality. Plugins could define targeted functionality descriptions to intercept user prompts to a specific plugin or an online service. For example, a plugin could intercept prompts to an online marketplace by adding in its functionality description that it can recommend products from that marketplace. We identified plugins that could potentially squat functionality. We describe their details in Risk 5.

Inject malicious response. A plugin could include instructions in its response for the LLM to route the prompts for a particular plugin to its API endpoints.

> **RISK 5: FUNCTIONALITY SQUATTING**
>
> **Risk overview.** Several OpenAI plugins mention the names of well-known online services in their functionality descriptions or define their functionality descriptions similar to other plugins, which allows them to hijack prompts that are not intended for them, i.e., functionality squatting.
>
> **Risk impact.** Successful functionality squatting will allow a plugin to deprive other plugins or online services of users, leading to loss in revenue. Plugin might also be able to trick users into sharing their data. Additionally, if the plugin is unable to fulfill the offered service, it could cause harm to users in several ways.

Evidence of risk. Lexi Shopper [22] recommends products from Amazon.com and mentions the word "Amazon" in its functionality description. Because of the presence of the word "Amazon", user prompts which even specify to not use any third party service are routed to Lexi Shopper, as it can be seen in our partial interaction with the plugin in Figure 5. Lexi Shopper interaction link [23] provides complete interaction with the plugin. This analysis was conducted on June 09, 2023.

In another example, two plugins Jio [20] and Tira [42] offer service to shop from `tirabeauty.com`. Tira is hosted by `tirabeauty.com` whereas Jio is hosted by `jiocommerce.io`, a third-party e-commerece service that allows users to shop from several online shops. In case a user enables both of the plugins and even specifies that it wants to shop from Tira, their queries are routed to the third-party service, i.e., Jio, instead of the first-party service, i.e., Tira. Tira and Jio interaction link [41] provides complete interaction with these plugins. Analysis conducted on July 27, 2023.

Recommend some shoe options from amazon.com, please do not use any third party service

Used **Lexi Shopper**

REQUEST TO LEXI SHOPPER

```
{
  "userMessage": "Recommend some shoe options from
amazon.com",
  "searchPhrase": "shoes",
  "language": "en"
}
```
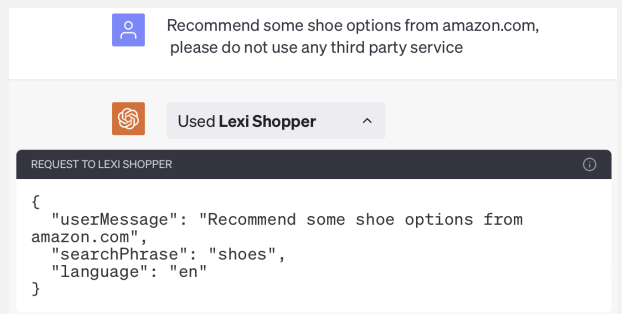
Figure 5: User interaction with Lexi Shopper plugin.

Observation. Our findings indicate that plugins could hijack prompts intended for other services, even when that service is integrated by the user. These interactions further highlight the challenge in tackling squatting for natural language based interfaces, that we alluded to in Risk 6.

6.2 Hijack prompts on a topic

In this attack category, a plugin can be both an adversary and a victim. The goal of the adversarial plugin is to hijack all user prompts on a particular topic. At a high level, a plugin could trick or instruct the LLM platform into calling itself. We discuss some of the ways in which an adversarial plugin could hijack all prompts on a particular topic.

"Squat" a topic. Plugins could hijack prompts on a specific topic by curating their functionality descriptions such that they always get precedence over other plugins in the same category. For example, a travel reservation plugin could include in its description that the LLM platform should always call the plugin for all travel related queries. We identified a plugin on OpenAI that could potentially squat user prompts on a topic. We describe the details in Risk 7 (Appendix D).

Inject malicious response. Similar to including instructions in its functionality description, a plugin could instruct the LLM platform via its response to always send user prompts on a particular topic to the plugin.

6.3 Influence prompts to another plugin

In this attack category, an attacker's goal is to influence the prompts to another plugin. Examples of influence could include altering the data sent to the another plugin, similar to a man-in-the-middle attack, or triggering another plugin, to launch a denial-of-service attack. At a high level, an attacker would need to trick the LLM platform to launch this attack. We describe a potential mechanism through which a plugin could manipulate the transmission of data to another plugin.

Exploit multipart prompts. A plugin service could exploit the workflow of multipart user requests, where several plugins interact with each other to resolve a user request. For example, a plugin could include altered data or a malicious payload in response that will be sent as an input to another plugin.

7 Discussion and conclusion

7.1 Exacerbation of NLP-related challenges

While many of the issues that we identified are echoes of the challenges in securing previous platforms (e.g., smartphones, IoT), the complexity of natural language is one of the more unique aspects and fundamental challenges in securing LLM-based platforms. In the plugin-integrated platforms we considered, natural language is used (1) by users to interact with the platform and plugins, (2) by the platform and plugins to interact with users, and (3) even by plugins to interact with the platform (e.g., through functionality descriptions) and other plugins (e.g., through instructions in API responses). Potential ambiguity and imprecision in the interpretation of natural language, as well as the application of policies to natural language, can create challenges in all of these interactions.

7.1.1 Interpretation of functionality in natural language

In conventional computing platforms, applications define their functionality through constrained programming languages without any ambiguity. In contrast, LLM platform plugins define their functionality through natural language, which can have ambiguous interpretations. For example, the LLM platform may in some cases interpret the functionality too broadly, or too narrowly, both of which could cause problems (see Risks 5 and 7 as examples). Interpreting language also requires contextual awareness, i.e., plugin instructions may need to be interpreted differently in different contexts. For example, it might be okay for the LLM platform to behave a certain way while a user interacts with a plugin, but not okay to persist with that behavior when the plugin is not in use (see Risk 3 as an example). In summary, the key challenge for LLM platforms is to interpret plugin functionality so as to not cause ambiguity, or in other words, LLM platforms must

figure out mechanisms that allow them to interpret functionality similarly to the unambiguous (or, much less ambiguous) interpretation in other computing platforms.

7.1.2 Application of policies on natural language content

Even if LLM platforms can precisely interpret the functionality defined in natural language or if functionality is precisely defined through some other means, it will still be challenging to apply policies (e.g., content moderation) over the natural language content returned by users, plugins, or within the LLM platform. For example, there may be a mismatch between the interpretation of the policy by the LLM platform, users, and plugins, e.g., on what is considered personal information (by building on attacks in 4.3, of which Appendix E.1 discusses an example). Similarly, in instances where there is a contradiction between the policies specified by the plugin or between the policies specified by the user and the plugin, the LLM platform would need to make a preference to resolve the deadlock, which may not be in favor of users. An LLM platform may also not apply the policies retrospectively, which may diminish its impact. For example, a policy that specifies that no personal data needs to be collected or shared may not apply to already collected data (by building on attacks in 4.3 of which Appendix E.1.1 discusses an example).

7.2 Towards secure LLM-based platforms

Stepping back from NLP-specific challenges to LLM-based platforms as a whole, we emphasize that security, privacy, and safety should be key considerations in the design process.

The restrictions and suggestions provided by LLM platforms (discussed in Section 2.3) are a step in the right direction, but they are insufficient to secure LLM platforms. We recommend that LLM platform designers consider security, privacy, and safety — e.g., by applying our framework — *early* in the design of their platforms, to avoid situations in which addressing issues later requires fundamental changes to the platform's architecture. The systemic nature of our findings and examples of attack potentials suggests that perhaps such a process was not used in the design of ChatGPT plugin ecosystem. In many cases, defensive approaches do not need to be invented from scratch: LLM platform designers can take inspiration from several sources, including from well-established practices to guard against known attacks, by repeating the threat modeling that we did in this paper, and by building on the security principles defined by prior research, such as by Saltzer and Schroeder [102].

We elaborate now on possible practical approaches for securing LLM platforms that wish to integrate untrusted third parties (e.g., plugins), and then step back to consider the potential future of LLM-based platforms more generally.

7.2.1 Anticipating and mitigating malicious third parties

A core issue underlying many of the risks we discussed is that third-party plugins may be malicious or buggy — an issue familiar to us from many past platforms [56, 57, 59, 74]. At the highest level, LLM platforms that want to integrate plugins should minimize trust in these third parties and design the platform to manage any potential risk. There is significant precedent in other platforms that can provide design inspiration to LLM platform creators.

For example, to ensure that plugin behavior does not change at runtime and that LLM platforms get an opportunity to review plugin code on each update, LLM platforms could host the plugin source code instead of plugin developers (elaborated on in Appendix A.1), similar to established platforms, such as mobile and web. Another avenue is to technically limit the functionality exposed to plugins. For example, LLM platforms could enforce a permission model, similar to mobile platforms, to regulate access to data and system resources.

Another strategy to minimize the impact of a problematic plugin is to isolate plugin execution from that of other plugins or the rest of the system, e.g., similar to site isolation in browsers through sandboxes [101]. At present (on the OpenAI platform we tested), all plugins execute together in the context of the same conversation. On the one hand, this execution model allows plugins to synergize well with each other, but on the other hand it exposes user interactions with one plugin to another. LLM platforms still could support plugin interaction and eliminate unnecessary data exposure by running each plugin in a sandbox and by clearly defining a protocol for sharing information across sandboxes, similar to cross-document messaging on the web [109].

In addition, LLM platforms should clearly state and enforce their policies and guidelines for plugin behavior, which may not currently be the case (e.g., see Appendix G).

7.2.2 Anticipating future LLM-based systems

Looking ahead, we can and should anticipate that LLMs will be integrated into other types of platforms as well, and that the plugin-integrated LLM chatbots of today are early indicators of the types of issues that might arise in the future. For example, we can anticipate that LLMs will be integrated into voice assistant platforms (such as Amazon Alexa), which already support third-party components ("skills", for Alexa). Recent work in robotics has also integrated LLMs into a "vision-language-action" model in which an LLM directly provides commands to a physical robot [50]. Future users may even interact with their desktop or mobile operating systems via deeply-integrated LLMs. In all of these cases, the NLP-related challenges with the imprecision of natural language, coupled with the potential risks from untrustworthy third parties, physical world actuation, and more, will raise serious potential concerns if not proactively considered. The designers of future LLM-based computing platforms should architect their platforms to support security, privacy, and safety early, rather than attempting to retroactively address issues later.

Acknowledgements

References

[1] Ai agents. https://ai-agents-plugin.vercel.ap.

[2] Ai agents interaction link. https://github.com/llmps/pa/blob/main/aiagents-interaction.pdf.

[3] Amzpro. https://turboooo.com/.

[4] Amzpro interaction link. https://github.com/llmps/pa/blob/main/amzpro-interaction.pdf.

[5] Askyourpdf plugin. https://askyourpdf.com.

[6] Askyourpdf privacy policy. AskYourPDF_privacypolicyhttps://web.archive.org/web/2023060807 4506/https://askyourpdf.com/terms.

[7] Autoinfra1 interaction link. https://github.com/llmps/pa/blob/main/autoInfra1-interaction.pdf.

[8] Chatgpt other language interaction. https://github.com/llmps/pa/blob/main/other-language-interaction.pdf.

[9] Chatgpt ssh plugin. https://chatsshplug.com.

[10] Chatsshplug interaction link. https://github.com/llmps/pa/blob/main/chatsshplug-interaction.pdf.

[11] Clinical trial radar. trialradar.marketflare.repl.co.

[12] Clinical trial radar interaction link. https://github.com/llmps/pa/blob/main/clinicaltrialradar-interaction.pdf.

[13] Creatuity stores. https://creatuity.com.

[14] edx plugin. https://www.edx.org/.

[15] Expedia. https://www.expedia.com.

[16] Expedia, trip.com, and klook interaction link. https://github.com/llmps/pa/blob/main/expedia-trip-klook-interaction.pdf.

[17] It's all fun and game until you have to personally test 100's of chatgpt plugins. https://twitter.com/OfficialLoganK/status/1659729951804161536.

[18] Jiggybase plugin. https://jiggybase.plugin.jiggy.ai.

[19] Jiggybase privacy policy. https://jiggy.ai/legal.

[20] Jio copilot. https://www.jiocommerce.io/co-pilot.

[21] Klook. https://www.klook.com.

[22] Lexi shopper. https://lexi-shopping-assistant-chatgpt-plugin.iamnazzty.repl.co.

[23] Lexi shopper interaction link. https://github.com/llmps/pa/blob/main/lexishopper-interaction.pdf.

[24] Magic after prompting link. https://github.com/llmps/pa/blob/main/magic-interaction-2.pdf.

[25] Magic before prompting link. https://github.com/llmps/pa/blob/main/magic-interaction.pdf.

[26] Magic plugin. https://getmagic.com.

[27] Pdf exporter - a chatgpt plugin. https://www.replypdf.com/.

[28] Pdf exporter - terms of use. https://web.archive.org/web/20230608205302/https://www.replypdf.com/legal.

[29] Pdf exporter interaction link. https://github.com/llmps/pa/blob/main/chatsshplug-pdfexporter-interaction.pdf.

[30] Playlistai. https://playlistai-plugin.vercel.app.

[31] Playlistai interaction link. https://github.com/llmps/pa/blob/main/playlistai-interaction.pdf.

[32] Playlistai stripe payment link. https://buy.stripe.com/00gcN073P1xV0SY5kn?prefilled_email=rmfmcbqrfk%40privaterelay.appleid.com.

[33] Reflect app privacy policy. https://web.archive.org/web/20230318175219/https://reflect.app/terms.

[34] Reflect notes interaction link. `https://github.com/llmps/pa/blob/main/reflectnotes-interaction.pdf`.

[35] Resumecopilot plugin. `https://c-resume.copilot.us`.

[36] Resumecopilot privacy policy. `https://web.archive.org/web/20230610224109/https://c-resume.copilot.us/home/terms`.

[37] Sakenowa.

[38] Scraper plugin. `https://gafo.tech/`.

[39] Seo plugin. `seo-plugin.orrenpruckun.com`.

[40] Think better with reflect. `https://reflect.app/`.

[41] Tira and jio interaction link. `https://github.com/llmps/pa/blob/main/tira-jio-interaction.pdf`.

[42] Tira beauty. `https://www.tirabeauty.com`.

[43] Tira interaction link. `https://github.com/llmps/pa/blob/main/tira-interaction.pdf`.

[44] Trip.com. `https://www.trip.com`.

[45] Uniket and tira interaction link. `https://github.com/llmps/pa/blob/main/tira-uniket-interaction.pdf`.

[46] Uniket beauty. `https://www.uniket.store`.

[47] Welcome to upskillr.ai. `https://www.upskillr.ai/`.

[48] autoinfra.ai. Automate your devops + infra. `autoinfra.ai`.

[49] Eugene Bagdasaryan, Tsung-Yin Hsieh, Ben Nassi, and Vitaly Shmatikov. (ab) using images and sounds for indirect instruction injection in multi-modal llms. *arXiv preprint arXiv:2307.10490*, 2023.

[50] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar

Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control. arXiv:2307.15818, 2023.

[51] Yunang Chen, Yue Gao, Nick Ceccio, Rahul Chatterjee, Kassem Fawaz, and Earlence Fernandes. Experimental security analysis of the app model in business collaboration platforms. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[52] Camille Cobb, Milijana Surbatovich, Anna Kawakami, Mahmood Sharif, Lujo Bauer, Anupam Das, and Limin Jia. How risky are real users' ifttt applets? In *USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2020.

[53] Federal Trade Commission et al. Data brokers: A call for transparency and accountability. *Washington, DC*, 2014.

[54] Scott A Crosby and Dan S Wallach. Denial of service via algorithmic complexity attacks. In *12th USENIX Security Symposium (USENIX Security 03)*, 2003.

[55] Lavina Daryanani. How to jailbreak chatgpt. `https://watcher.guru/news/how-to-jailbreak-chatgpt`.

[56] William Enck, Damien Octeau, Patrick D McDaniel, and Swarat Chaudhuri. A study of android application security. In *USENIX security symposium*, 2011.

[57] Shehroze Farooqi, Maaz Musa, Zubair Shafiq, and Fareed Zaffar. Canarytrap: Detecting data misuse by third-party apps on online social networks. *Proceedings on Privacy Enhancing Technologies*, 2020.

[58] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[59] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

[60] Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich, and Jason Polakis. O single {Sign-Off}, where art thou? an empirical analysis of single {Sign-On} account hijacking and session management on the web. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1475–1492, 2018.

[61] Google. Google bard. `https://bard.google.com/`.

[62] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. *arXiv preprint arXiv:2302.12173*, 2023.

[63] Arjun Guha, Matthew Fredrikson, Benjamin Livshits, and Nikhil Swamy. Verified security for browser extensions. In *2011 IEEE Symposium on Security and Privacy*, 2011.

[64] Umar Iqbal, Pouneh Nikkhah Bahrami, Rahmadi Trimananda, Hao Cui, Alexander Gamero-Garrido, Daniel Dubois, David Choffnes, Athina Markopoulou, Franziska Roesner, and Zubair Shafiq. Tracking, profiling, and ad targeting in the alexa echo smart speaker ecosystem. In *ACM Internet Measurement Conference (IMC)*, 2023.

[65] Maurice Jakesch, Advait Bhat, Daniel Buschek, Lior Zalmanson, and Mor Naaman. Co-writing with opinionated language models affects users' views. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023.

[66] The Wall Street Journal. Amazon changed search algorithm in ways that boost its own products. `https://themarkup.org/google-the-giant/2020/07/28/google-search-results-prioritize-google-products-over-competitors`.

[67] Daniel Kang, Xuechen Li, Ion Stoica, Carlos Guestrin, Matei Zaharia, and Tatsunori Hashimoto. Exploiting programmatic behavior of llms: Dual-use through standard security attacks. *arXiv preprint arXiv:2302.05733*, 2023.

[68] Tadayoshi Kohno, Yasemin Acar, and Wulf Loh. Ethical frameworks and computer security trolley problems: Foundations for conversations. In *USENIX Security*, 2023.

[69] Chieh-Jan Mike Liang, Börje F Karlsson, Nicholas D Lane, Feng Zhao, Junbei Zhang, Zheyi Pan, Zhao Li, and Yong Yu. Sift: building an internet of safe things. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, pages 298–309, 2015.

[70] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[71] Lei Liu, Xinwen Zhang, Guanhua Yan, Songqing Chen, et al. Chrome extensions: Threat analysis and countermeasures. In *NDSS*. Citeseer, 2012.

[72] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[73] The Markup. Google's top search result? surprise! it's google. `https://themarkup.org/google-the-giant/2020/07/28/google-search-results-prioritize-google-products-over-competitors`.

[74] Jonathan R. Mayer and John C. Mitchell. Third-party web tracking: Policy and technology. In *2012 IEEE Symposium on Security and Privacy*, 2012.

[75] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

[76] Lukasz Olejnik, Tran Minh-Dung, and Claude Castelluccia. Selling off privacy at auction. In *Network and Distributed System Security Symposium*, 2014.

[77] OpenAI. Brand guidelines. `https://openai.com/brand#plugins`.

[78] OpenAI. Can i charge people money for my plugin? `https://platform.openai.com/docs/plugins/production/can-i-charge-people-money-for-my-plugin`.

[79] OpenAI. Chatgpt plugins. `https://openai.com/blog/chatgpt-plugins`.

[80] OpenAI. Chatgpt plugins documentation. `https://platform.openai.com/docs/plugins/introduction`.

[81] OpenAI. Data controls faq. `https://help.openai.com/en/articles/7730893-data-controls-faq`.

[82] OpenAI. Domain verification and security. `https://platform.openai.com/docs/plugins/production/domain-verification-and-security`.

[83] OpenAI. Getting started. `https://platform.openai.com/docs/plugins/getting-started`.

[84] OpenAI. Gpt-4 is openai's most advanced system, producing safer and more useful responses. `https://openai.com/gpt-4`.

[85] OpenAI. Introducing chatgpt. `https://openai.com/blog/chatgpt`.

[86] OpenAI. Ip egress ranges. `https://platform.openai.com/docs/plugins/production/ip-egress-ranges`.

[87] OpenAI. Model behavior feedback. `https://openai.com/form/model-behavior-feedback`.

[88] OpenAI. Plugin authentication. `https://platform.openai.com/docs/plugins/authentication`.

[89] OpenAI. Plugin policies. `https://openai.com/policies/usage-policies#plugin-policies`.

[90] OpenAI. Plugin store – submit a plugin for review. `https://platform.openai.com/docs/plugins/review/plugin-store`.

[91] OpenAI. Plugin store review criteria. `https://platform.openai.com/docs/plugins/review/plugin-store`.

[92] OpenAI. Plugin terms. `https://openai.com/policies/plugin-terms`.

[93] OpenAI. Rate limits. `https://platform.openai.com/docs/plugins/production/rate-limits`,.

[94] OpenAI. Updating your plugin. `https://platform.openai.com/docs/plugins/production/updating-your-plugin`.

[95] OpenAI. What makes a great plugin. `https://platform.openai.com/docs/plugins/review/what-makes-a-great-plugin`.

[96] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos P. Markatos. Cookie synchronization: Everything you always wanted to know but were afraid to ask. In *The Web Conference (WWW)*, 2019.

[97] Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*, 2022.

[98] Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. In *NeurIPS ML Safety Workshop*, 2022.

[99] Johann Rehberger. Indirect prompt injection via youtube transcripts. `https://embracethered.com/blog/posts/2023/chatgpt-plugin-youtube-indirect-prompt-injection/`.

[100] Johann Rehberger. Plugin vulnerabilities: Visit a website and have your source code stolen. `https://embracethered.com/blog/posts/2023/chatgpt-plugin-vulns-chat-with-code/`.

[101] Charles Reis, Alexander Moshchuk, and Nasko Oskov. Site isolation: Process separation for web sites within the browser. In *28th USENIX Security Symposium (USENIX Security 19)*, 2019.

[102] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 1975.

[103] Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. Extension breakdown: Security analysis of browsers extension resources control policies. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017.

[104] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[105] Dolière Francis Somé. Empoweb: Empowering web applications with browser extensions. In *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[106] TechCrunch. Google launches a smarter bard. `https://techcrunch.com/2023/05/10/google-launches-a-smarter-bard/`.

[107] TheVerge Tom Warren. These are microsoft's bing ai secret rules and why it says it's named sydney. `https://www.theverge.com/23599441/microsoft-bing-ai-sydney-secret-rules`.

[108] Giridhari Venkatadri, Piotr Sapiezynski, Elissa M. Redmiles, Alan Mislove, Oana Goga, Michelle Mazurek, and Krishna P. Gummadi. Auditing offline data brokers via facebook's advertising platform. In *The World Wide Web Conference*, 2019.

[109] WHATWG. Cross-document messaging. `https://html.spec.whatwg.org/multipage/web-messaging.html`.

[110] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2023.

[111] Andy Zou, Zifan Wang, , J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023.

A Detailed responsibilities of key stakeholders

A.1 Plugins

1. **Development & updates.** Plugin developers are responsible for complete end-to-end development of their plugins, includin defining plugin functionality, exposing API end points, and managing and updating their plugins over time.

2. **Hosting & availability.** Currently, in contrast to more established platforms (e.g., mobile), in LLM platforms

plugin vendors are responsible for hosting plugins entirely on their own servers. Plugin developers can make their plugins available to users both through plugin store(s) and outside plugin stores.

3. **LLM authentication.** Plugin developers can optionally provide authentication information so that LLMs can send them authenticated requests. Plugins can also explicitly allow API endpoint network traffic only from LLM platforms and restrict all other traffic.

4. **User authentication.** Plugins can setup optional user-level authentication through custom UIs and single sign-on services, such as OAuth [88].

5. **Prompt resolution.** Plugins are responsible for sending the data requested by LLMs through API requests and successfully fulfilling the commands sent to them by LLMs. Plugins also possess full control and autonomy over fulfilling the commands, and data sent along with them, by the LLMs.

A.2 LLM platform

1. **Code review, availability, and updates.** LLM platforms review plugins as per plugin store guidelines [90], and make them available on the plugin store. Furthermore, if plugins update, LLMs are responsible for reviewing the updated manifests and specification, which are hosted on plugin-controlled servers.

2. **Providing user authentication interfaces.** LLM platforms are also responsible for providing interfaces to plugins so that they can offer, both custom and single sign-on based, authentication to users.

3. **Initiating plugins.** LLM platforms load the plugin functionality and API descriptions to build the necessary context for using plugins. LLMs are also responsible for identifying the correct plugin and API call, and for sending the correct data in the required format to the API endpoint. In case of multiple plugins that offer the same functionality, LLM calls the plugin whose functionality description *best matches* the user command or all of the relevant plugins.

4. **Facilitating user-plugin interaction.** LLMs resolve user command that requires the use of plugins. LLMs are responsible for formatting the data received from API endpoints and displaying the output to the user. Based on the user prompt and the plugin output, LLMs may call additional plugins to completely resolve the user request. Note that all communication between LLM platforms and plugins is server-to-server and only end results are shown to the user. LLMs also do not possess any control over what data is sent to them by plugins or over the successful and responsible resolution of command sent to the plugin API endpoint.

A.3 Users

1. **Plugin installation & deletion.** Users install plugins from the plugin store or other unvetted sources, currently without any recommendations from LLMs. Users are also responsible for deleting the plugins.

2. **Account management.** Users can create accounts on third-party plugin platforms or sign in to them on LLM platforms, through single sign-on services.

3. **Interacting with plugins.** Users give prompts to LLMs that require the use of the plugin, with or without specifying the name of the plugin. Users must sometimes provide additional information to LLM that it might need to communicate to the plugin to successfully resolve user request. Users are also responsible for acting on the final recommendations from plugins, if any.

B Detailed discussion of ethics and disclosure

As noted in Section 3.4, in evaluating the ethics and morality of this research, we drew from both consequentialist and deontological traditions [68]. We first present our consequentialist analysis. LLM and LLM-based systems are innovating incredibly rapidly, and researchers (including ourselves) are uncovering vulnerabilities with deployed systems.

The first question we asked ourselves: is it ethical and moral to develop and share an attack taxonomy for LLM-based plugin systems? (Just developing but not sharing would have little consequentialist output). We determined that the benefits of creating and sharing such a taxonomy outweigh the harms. The taxonomy can enable those developing LLM-based systems and the security research community to have dedicated, detailed discussions about how to mitigate future vulnerabilities, as well as discussions about which vulnerabilities may be high risk and which may not. Further, as is often the case in other technology sub-areas, those seeking to cause harm to platforms and users (the adversaries) may be developing attack capabilities in silent. If those adversaries manifest before platforms and the security community are able to proactively discuss and develop defenses, there could be significant harms.

The next question we asked ourselves was: is it ethical and moral to evaluate a real system (ChatGPT) with respect to our attack taxonomy? Here, we observe several key benefits. First, our attack taxonomy benefited greatly from the proactive experimentation with ChatGPT and the resulting lessons from such experimentation and, hence, the experimentation with ChatGPT contributed to the benefits described in the above paragraph. Second, we again observe that adversaries may be operating silently and, hence, it is beneficial to understand actual risks before adversaries manifest. Third, we stress that we did *not* mount any attacks against any parties.

Rather, we studied what different plugins *could* do *if* they were adversarial.

As part of our research process, we also asked ourselves: is it ethical and moral to experimentally develop "malicious" (or at least plugins with the potential to be malicious though perhaps reserved in some way, e.g., collect private information but not use it) and publish them on the OpenAI's plugin store? A benefit of doing so would be a concrete experimental evaluation of OpenAI's review process. However, the harms in this scenario — the potential to accidentally harm users even if we took precautions — outweigh the benefits, especially since information about the OpenAI's plugin review process (a single reviewer assigned to review all published plugins [17]) is already known. Hence, we did not seek to publish any plugins on the OpenAI's plugin store.

The next question we asked ourselves was: what should be the disclosure process? A consequentialist analysis of the disclosure processes is complicated by the significant uncertainties about the future of LLMs, LLM-based systems, and adversarial capabilities. It is known that consequentialist analyses under uncertainties can be fundamentally challenging [68]. Our deontological analysis, however, did lead to a clear and conservative conclusion. Hence, we center our deontological analysis in this discussion. Specifically, we observe that the people running OpenAI have rights. In this case, we believe that they have the right to learn our findings and have a chance to respond prior to our paper being public. Hence, we have determined that the morally correct process is to share our findings with OpenAI before publishing this paper, which we have already done.

Another question we asked ourselves: what should be the process of disclosing our findings to any plugins that we mention in this paper? As noted elsewhere, it bears re-stressing that we did *not* seek to find vulnerabilities in plugins and we did *not* attack any plugins. Rather, we used properties of existing plugins to gather evidence about the potential capabilities of adversarial plugins. Still, from a deontological perspective, we determined that plugin authors have the right to know about our analyses, and hence we have informed plugin authors about our results and findings with respect to their plugins.

C Example of a potential plugin squatting risk

Building on the attack described in Section 4.2 (*"Squat" another plugin*), we identified plugins that could be squatting other plugins. We describe the details in Risk 6.

RISK 6: PLUGIN SQUATTING

Risk overview. OpenAI hosts several plugin pairs with identical names, manifests, and API specifications. Having an identical code base allows plugins to impersonate or "squat" other plugins.

Risk impact. Users could install squatted plugins, assuming that they are installing the genuine plugins and share their data. Even in instances where the duplicate plugins are benign, they could mistakenly hijack prompts and data intended for the original plugin. Squatted plugins could also cause reputational and monetary damage to the original plugins.

Evidence of risk. We found two instances of duplicate plugins, i.e., Upskillr [47] and Scraper [38]. Figure 6 shows the duplicate presence of Upskillr on the OpenAI plugin store. We analyzed the manifest and specification of duplicate plugins and found that they have the same code base but are hosted by distinct developers, i.e., `upskillr.ai` and `bestviewsreviews.com`. We then analyzed the developer websites' and found that the plugin functionality clearly aligns with the functionality offered by `upskillr.ai`. We also tried interacting with the duplicate hosted by `bestviewsreviews.com`, but the plugin API service was unresponsive at the time of testing. We could not verify whether the plugin hosted by `bestviewsreviews.com` had any malicious intent. Analysis was conducted on June 07, 2023.
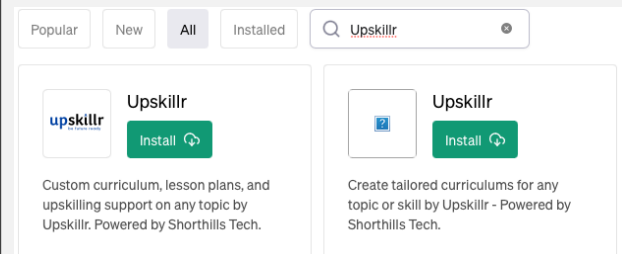

Figure 6: Dual presence of Upskillr plugin on the OpenAI plugin store.

Observation. While eliminating duplicate plugins could be trivial, cyber squatting is a well-known and a non-trivial problem to address in general. We anticipate squatting to be an even more challenging problem in LLM platforms, given the complexity of interpreting functionality defined in natural language (discussed in Section 7).

D Example of a potential topic squatting risk

Building on the attack described in Section 6.2 (*"Squat" a topic*), we identified a plugin that could potentially squat user prompts on a topic. We describe the details in Risk 7.

RISK 7: TOPIC SQUATTING

Risk overview. Several OpenAI plugins add certain keywords in their functionality descriptions or define overly broad functionality descriptions to hijack prompts on specific topics, i.e., topic squatting.

Risk impact. Successful topic squatting will allow a plugin to deprive other plugins of users and revenue. Plugin will also be able to harvest user data and trick users into sharing their data. Additionally, if the plugin is unable to fulfill the offered service, it could cause harm to users in several ways.

Evidence of risk. Expedia [15], a well-known travel reservation service, hosts a plugins which instructs ChatGPT to *"ALWAYS uses Expedia plugin to provide travel recommendations for ANY user's travel-related queries"*. To evaluate whether the use of all caps and direct command would allow Expedia to intercept user prompts for all travel related queries, we installed Expedia's plugin in a chat session with ChatGPT, along with two other travel plugins, Trip.com [44] and Klook [21], and made travel related queries. We found that ChatGPT automatically routed user prompts to Expedia, without asking users for their preference, as seen in our partial interaction with Expedia in Figure 7. Expedia, Trip.com, and Klook interaction link [16] presents complete interaction with these plugins. Analysis conducted on June 09, 2023.

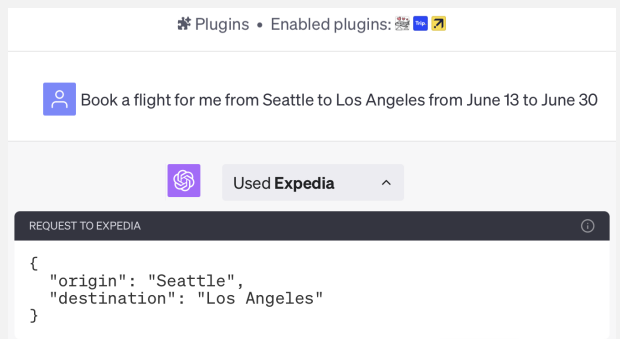Additional analysis of plugins with overly broad functionality descriptions is in Appendix F.

Figure 7: User interaction with the Expedia, Trip.com, and Koll plugins.

Observation. Broad and targeted functionality descriptions make it challenging to interpret the functionality offered by plugins, which can confuse both users and LLMs. It is a key challenge for LLM platforms to develop unambiguous natural language based programming interfaces.

E   Data exfiltration

We encountered several plugins which collect excessive amount of user data, including personal and sensitive data.

E.1   Personal and personally identifiable information collection

One such plugin is Clinical Trial Radar [11], which assists users in finding and understanding clinical trials for various diseases. To recommend relevant trials, the plugins collects sensitive and personal user information, including the diseases they are suffering, disease stage, prior treatments, location, and other health details. The plugins instructs the LLM, through its description, to limit user data collection and anonymize user input. However, even with these filters, ChatGPT does not warn users to not share excessive data. We also did not notice any attempt from ChatGPT to anonymize user input. Clinical Trial Radar interaction link [12] presents our interaction with the plugin. This analysis was conducted on June 07, 2023.

E.1.1   Sharing of already collected information

Another similar plugin called Magic [26] had even more specific instructions for ChatGPT to *"Keep in mind that you do not need to ask for personal information such as users' name or email address"*. However, based on our interactions we noticed that these instructions did not apply to the data that was shared before prompting the plugin. Specifically, we provided personal details (such as name, email address, date of birth, health issues) to ChatGPT and asked it to arrange the travel. When ChatGPT contacted Magic, these details were automatically shared in the API request. Magic before prompting link [25] provides interaction with the plugin when data was shared before prompting. Magic after prompting link [24] provides interaction with the plugin when it was prompted before sharing any data. This analysis was conducted on July 31, 2023.

E.2   Document exfiltration

Another class of plugins, directly asks users to upload documents, such as resumes and emails. ResumeCopilot [35] is one such plugin which asks the users to upload their resume by providing them an external link. Once users upload their resume, it instructs ChatGPT to improve it by collaborating with the user. Another similar plugin AskYourPDF [5] asks users to upload any PDF document and offers to search information in the document based on user queries. JiggyBase [18], another plugin goes a step further and allows user to upload collections of documents, such as web pages and emails, to find answers to questions and retrieve relevant information. We also analyzed privacy polices of all these plugins. ResumeCopilot and AskYourPDF do not provide any details about how they use user data and especially if they use this data for purposes that go beyond providing users the functionality [6, 36]. JiggyBase, however, states that the user content is only used for functional purposes [19]. In addition, the plugin promises that the users chat messages and data will not be used by OpenAI for model training. It is unclear if plugins can hold such promises, since user chat history is by default recorded by OpenAI to improve and train LLMs [81].

F   Plugins with broad API description

We encountered several plugins with overly broad or vague descriptions. This was especially the case for plugins for on-line marketplaces, which in many instances did not specify the products or product segments that they sell on their marketplaces. For example, Creatuity Stores [13] describes in its functionality that it can be used to search for products for given description in all on-line stores integrated with the plugin, without specifying which stores are integrated with the plugin. Similarly, Tira [42] describes to the LLM that it can be used to to search for products within Tira's marketplace, without specifying that it only sells beauty products. Note that Tira clearly mentions in its *human_description* that it sells beauty products (but not in *model_description*). Upon interaction with Tira, we noted that shopping queries outside the scope of beauty products, such as groceries were routed to Tira because of its broad description. Our interaction can be viewed by visiting the Tira interaction link [43]. This analysis was conducted on June 06, 2023.

G   Policy violations

We encountered several plugins which potentially violate OpenAI plugin policies, recommendations, and guidelines [78, 89, 91]. For example, despite OpenAI currently prohibiting developers from charging people money for plugins, we encountered a plugin, called PlaylistAI [30], which requests money from users. PlaylistAI interaction link [31] presents our interaction with PlaylistAI. We conducted this analysis on June 07, 2023. Note that the plugin does not surreptitiously asks for payments, but in fact clearly mentions it in its functionality description: *"If the plugin returns a 429 status code, then the user needs to pay to continue using the plugin"*. Upon clicking the payment link, users are redirected to a Stripe payment page [32] which allows users to pay to subscribe to the service. With such payment flows, malicious plugin services could trick users into visiting malicious webpages to steal their credit card details.

OpenAI also prohibits plugins from including instructions for ChatGPT in plugin response messages [89]. This policy is also violated by several plugins, such as AI Agents [1]. AI Agents API JSON response contains a field called *instructions* for ChatGPT, to interpret the remaining fields in the JSON. Complete conversation with AI Agents can be viewed by visiting the AI Agents interaction link [2]. We conducted this analysis on June 09, 2023.

Similarly, OpenAI prohibits developers from using the words like *plugin*, *ChatGPT*, or *OpenAI* in the plugin name or description and requires that they use *correct grammar* in their functionality descriptions [91]. However, we found several plugins hosted on the OpenAI's plugin store which do not meet this criteria. SEO [39] is one such plugin, which makes spelling and grammatical mistakes in its functionality descrip-

tion. For example, "this" and "occurrence" are misspelled as "ths" and "occurance", respectively. Another plugins, called edX [14] mentions both *ChatGPT* and *plugin* in its description.