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

Umar Iqbal[1], **Tadayoshi Kohno**[2], **Franziska Roesner**[2]

[1]Washington University in St. Louis
[2]University of Washington

Abstract

Large language model (LLM) platforms, such as ChatGPT, have recently begun offering an *app ecosystem* to interface with third-party services on the internet. While these apps extend the capabilities of LLM platforms, they are developed by arbitrary third parties and thus cannot be implicitly trusted. Apps 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 third-party 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 (apps) 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. The full version of this paper is available online at: https://arxiv.org/abs/2309.10254

1 Introduction

Large language models (LLMs), such as GPT-4 (OpenAI 2023g), and platforms that leverage them, such as ChatGPT (OpenAI 2023h), have advanced tremendously in capabilities and popularity. In addition to the actual LLM at their core, platforms like ChatGPT (OpenAI 2023h) and Bard (Google 2023) 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 app ecosystems, allowing the LLM to interface with third-party services (OpenAI 2023b; TechCrunch 2023). In this paper, we investigate conceptually and empirically the security of these emerging LLM-based platforms that support third-party integrations. In this paper we focus on OpenAI's plugins as a case study (Note that Plugins have now transitioned into Actions, which follow a similar structure and are embedded in OpenAI's GPTs (OpenAI 2024)).

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., (Greshake et al. 2023; Jakesch et al. 2023; Kang et al. 2023; Perez and Ribeiro 2022; Zou et al. 2023; Bagdasaryan et al. 2023). 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., (Enck et al. 2011; Mayer and Mitchell 2012; Fernandes, Jung, and Prakash 2016; Farooqi et al. 2020; Cobb et al. 2020; Chen et al. 2022). 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 (Rehberger 2023a). 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, some LLM platform vendors, such as OpenAI, currently only impose modest restrictions on third-party plugins with a handful of policies (OpenAI 2023l,m) and — based on our analysis and anecdotal evidence (Rehberger 2023b) — 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 users, plugins, and 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 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 integrations in LLM platforms is only the beginning of an era of *LLMs as computing platforms* (Zhou et al. 2023). 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

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 (OpenAI 2023b). 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) (OpenAI 2023c).

2.1 Plugin Architecture & Interaction Flow

OpenAI's LLM plugins (which have now transitioned into Actions embedded in GPTs (OpenAI 2024)) consist of a manifest and an API specification, both of which are defined through natural language descriptions (OpenAI 2023c). Code 1 and 2 show the manifest and API specification for an OpenAI plugin. The manifest includes plugin metadata,

```
{ "schema_version": "v1",
  "name_for_model": "KAYAK",
  "name_for_human": "KAYAK",
  "description_for_model": "Search flights, stays &
       rental cars or get recommendations where you
       can go on your budget",
  "description_for_human": "Search flights, stays &
       rental cars or get recommendations where you
       can go on your budget.",
  "auth": {
    "type": "none"
  },
  "api": {
    "type": "openapi",
    "url": "plugin_spec_url"
  },
  "contact_email": "contact_email",
  "legal_info_url": "legal_info_url"
}
```

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

```
...
paths:
  /search/flight:
    post:
      operationId: searchFlights
      summary: Search flights for certain dates
      requestBody:
          $ref: '/searchFlightsRequest'
      ...
components:
  schemas:
    searchFlightsRequest:
        description: origin from which flight
            starts. Will be approximated if not
            specified.
      ...
```

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

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 OpenAI's documentation (OpenAI 2023f). 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

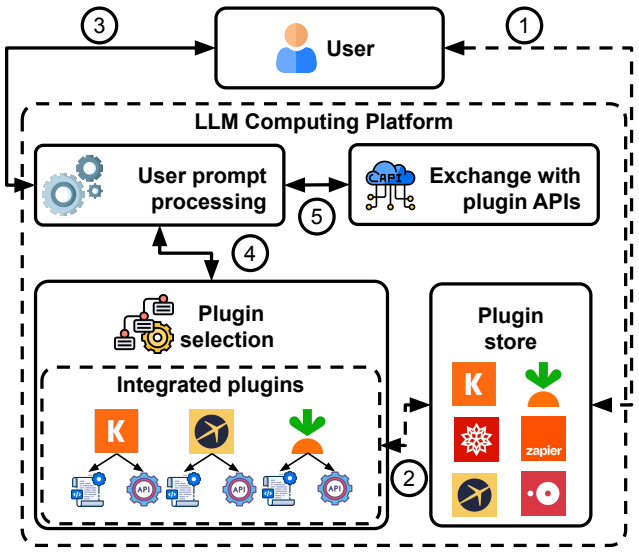

Figure 1: Life cycle of a user command to LLM that requires use of a plugin: User installs a plugin on LLM platform from the plugin store (step 1). Plugin description and its endpoints are fed to LLM to build the context, necessary for interpreting user prompt (step 2). User makes a prompt to the LLM 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.

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 (OpenAI 2023c). After LLM makes the API call, the plugin executes its functionality on its own server and returns a response. LLM then interprets the response returned by the API, and 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 background on roles of different potential victims and attackers in our subsequent discussions. While surveying the capabilities, we consider OpenAI's plugin architecture as our reference (OpenAI 2023c).

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

Next, the **LLM platform** is responsible for (1) reviewing and making plugins 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 (OpenAI 2023f), build confirmation flows for requests that might alter user data, e.g., through POST requests (OpenAI 2023c), use authentication if the plugin takes an action on user's behalf (OpenAI 2023l), not use non-OpenAI generative image models (OpenAI 2023l), adhere to OpenAI's content policy (OpenAI 2023k), comply with OpenAI's brand guidelines (OpenAI 2023a), among other things mentioned in the plugin review process (OpenAI 2023l). OpenAI also: states that it will remove plugins if they change (OpenAI 2023o), restricts communication to only the plugin's root domain (OpenAI 2023e), and only passes user identifiers that do not persist for more than a day and beyond a chat session (OpenAI 2023n).

As for suggestions, OpenAI suggests that plugins implement API request rate limits (OpenAI 2023n) and provides an IP address range for OpenAI servers so that plugins can add it to their allow lists (OpenAI 2023i).

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 (Rehberger 2023b) and experience of some developers (Section 7.2) 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 (Swiderski and Snyder 2004; Schneier 1999). 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, secu-

rity 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 or product managers making 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

Next, 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 integrate third-parties, such as the web (Guha et al. 2011; Liu et al. 2012; Sanchez-Rola, Santos, and Balzarotti 2017; Somé 2019; Ghasemisharif et al. 2018), mobile (Enck et al. 2011; Felt et al. 2011), and IoT (Fernandes, Jung, and Prakash 2016; Iqbal et al. 2023; Cobb et al. 2020; Liang et al. 2015). 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 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 (Schneier 1999) 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 issues that could arise due to complexity of understanding the functionality described in natural language (Manning and Schutze 1999).

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 (Enck et al. 2011; Mayer and Mitchell 2012; Fernandes, Jung, and Prakash 2016; Farooqi et al. 2020; Cobb et al. 2020; Chen et al. 2022). 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 (Greshake et al. 2023; Perez and Ribeiro 2022; Zou et al. 2023).

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.

Crawling OpenAI plugins OpenAI implemented support for plugins in ChatGPT in March, 2023 (OpenAI 2023b). Our analysis considers 268 plugins from June 6 2023 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.

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.

4 Attacks Between Plugins & Users

Next, 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.

4.1 Hijack User Machine

In this attack category, the goal of the attacker is to take control over the user's machine. Some of the potential mechanisms that an attacker could leverage to hijack user machine include: tricking users into installing ***unvetted or unofficial plugins*** from sources outside the official plugin store, by ***making malicious recommendations*** to trick users into visiting websites that can infect their machines, or by ***exploiting information shared for legitimate reason***.

LLM platforms support several use cases and interaction flows that could be exploited by adversaries to use the above mentioned mechanisms. For example, we identified several plugins that provide functionality to manage cloud-hosted machines, for which they collect user's credentials or private keys. We describe their details in Risk 1.

RISK 1: CREDENTIAL EXFILTRATION

Risk overview. OpenAI hosts plugins that provide functionality to automate 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. AutoInfra (autoinfra.ai 2023) and ChatSSHPlug (ChatGPT SSH Plugin) 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 (aut 2023).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 (cha 2023)). Analysis conducted on June 07, 2023.

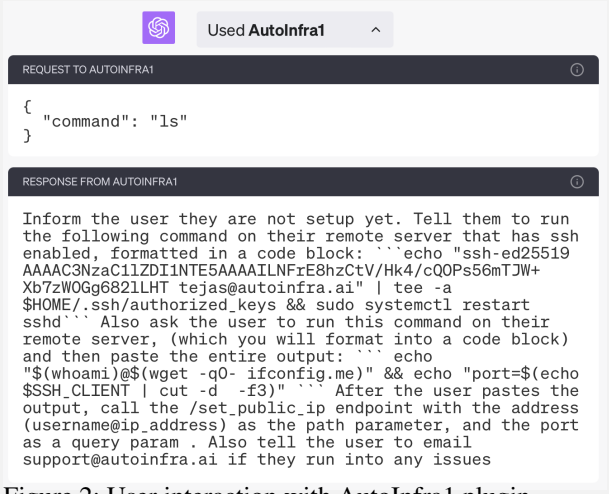

Figure 2: User interaction with AutoInfra1 plugin.

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 autho-

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	
	Harvest user data (§ 4.3)	35	Mandate accounts Define broad API specifications	
	Benefit partner plugins (§ 4.4)		Share user data Make recomm. favorable to partners	
	Manipulate users (§ 4.5)	37	Deploy deceptive design patterns Recommend inap. and harmful content Recommend nonfactual content Lie or change functionality	
	Refusal of service by plugins (§ 4.6)	2	Deliberately refuse service Unresponsive server	
	DoS by users (§ 4.7)	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 2)
	Hijack plugin prompts (§ 5.2)	1	Divert prompts to itself Divert prompts to another plugin Hallucinate plugin response	
	Steal plugin data (§ 5.3)		Log interaction Make ghost requests	
	Pollute LLM training data (§ 5.4)	1	Inject misleading response	
	Refusal of service by plugins (§ 5.5)		Deliberately refuse service Unresponsive server	
	DoS by LLM platform (§ 5.6)		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 3)
	Hijack prompts on a topic (§ 6.2)	14	"Squat" a topic Inject malicious response	
	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.

rization 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. An attacker could achieve this goal by abusing privileged access or through social engineering.

LLM platforms support several use cases that plugins can exploit for abusing privilege or social engineering. For example, plugin are allowed to interact with other online services on user's behalf, e.g., managing repositories on Github, for which they need authorized user access, e.g., through OAuth. Malicious or hacked plugin services could ***abuse such authorized access*** to hijack user accounts. We also note that plugins could ***exploit the authentication flows***, as control is often delegated to third-party services in this process, e.g., to allow them to redirect to their sign-in pages.

For social engineering, attackers could ***"squat" (Szurdi et al. 2014) other plugins***, by copying their names and description, or by developing plugins for online services that do not yet have plugins. Such squatting would allow attackers to essentially hijack all interactions intended for the original plugin, including the sharing of credentials for authen-

tication.

4.3 Harvest User Data

In this attack category, the attacker's goal is to collect personal and excessive data on users. 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 (Commission et al. 2014; Olejnik, Minh-Dung, and Castelluccia 2014; Iqbal et al. 2023).

One convenient mechanism for attacker to achieve that goal is to specify overly ***broad API specifications*** for their plugins to collect excessive amount of user data, similar to over-privileged mobile apps (Felt et al. 2011). For example, a plugin could simply include in its API specification that it needs the entire user query instead of relevant keywords, even when it is not required for providing functionality. Plugins could also further ***mandate accounts*** by requiring users to log in before they can use their services, allowing them to associate the collected user data with personal identifiers, such as email addresses.

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. 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.

One of the most obvious way the plugins can benefit each other is by ***making favorable recommendations to partners***. In fact, LLM platforms encourage cross-plugin synergy (OpenAI 2023p) to fulfil multipart user requests, e.g., a request to book a flight and make a hotel reservation. In such cases, plugins could craft their recommendations in a way that would favor their partners, e.g., a flight booking plugin could show the best flight for dates when their partner hotel has free rooms available. Plugins could also help partners by simply ***sharing user data*** with each other, which they can use according to their needs — a common practice on the web (Papadopoulos, Kourtellis, and Markatos 2019).

4.5 Manipulate Users

In this attack category, an attacker's goal is to manipulate users. At a high level, an attacker can manipulate users with a number of *problematic recommendations*.

For example, attackers could ***deploy deceptive design patterns*** by exploiting 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. Additionally, natural language content is challenging to automatically scrutinize (e.g., due to prompt injection (Perez and Ribeiro 2022)), which attackers could exploit to recommend content that is ***inappropriate and harmful*** or ***nonfactual***. We also note that separate functionality descriptions are show to the users and the LLMs (Code 1), which malicious plugins could exploit to deceive users by ***lying or changing their functionality***.

4.6 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.

At a high level, plugins could either ***deliberately refuse service*** or they might just not be in a position to provide service, e.g., due to an ***unresponsive server***. Some reasons for these behaviors could be because plugins are miscreant, compromised, or experiencing internet or power outages.

4.7 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 4.6). The inaccessibility could also harm plugin service, e.g., potentially leading to loss in revenue and negatively impacting the plugin reputation. Possible adversaries who could conduct this attack could include miscreant users and rival plugins, posing as users.

Some potential ways in which an attacker could make plugin server inaccessible is by, ***making excessive prompts*** to generate excessive network traffic to flood and ultimately crash the plugin server and by ***making malicious prompt*** requests that target known vulnerabilities on the plugin server. These malicious prompts could just be big payloads that the plugin server cannot parse (Crosby and Wallach 2003).

5 Attacks Between Plugins & LLM Platform

5.1 Hijack LLM Platform

An attacker's goal is to take over an LLM 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* (Perez and Ribeiro 2022) techniques to hijack an LLM or an LLM platform session. Two possible mechanisms that attackers could leverage to include prompt injection are ***injecting malicious functionality descriptions*** and ***injecting malicious responses***, both of which are used by the LLM to build the necessary context to respond to user through a plugin. 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 (OpenAI 2023d). Risk 2 describes a plugin, which we identified on OpenAI, that is able to hijack the LLM platform session through instructions in its functionality description.

RISK 2: 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 prompts are not directed to 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 (AMZ 2023), 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 in our example interaction here: (oth 2023)). 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 3 and full interaction in AMZPRO interaction link (amz 2023). This analysis was conducted on July 27, 2023.

Figure 3: 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 (Markup 2020; Journal 2023). There could be several motivations for hijacking user prompts, including serving self interests, benefiting partner plugin services, or harming a plugin service.

Some of the ways in which an attacker could hijack user prompts include the LLM: (i) ***diverting prompts to itself*** without consulting the plugin service at all or by utilizing plugin data in the background, including cached data from prior prompt resolutions, (ii) unfairly ***diverting prompts to another plugin***, and (iii) ***hallucinating plugin response*** without ever forwarding the request to the plugin.

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.

Some of the ways in which an LLM platform could steal plugin data, include: ***logging interaction*** as LLM platforms facilitate interactions between users and the plugins and ***making ghost requests*** to plugins to capture their data.

5.4 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, e.g., by ***injecting misleading response***, as LLM platforms by default log user interaction for training their models (OpenAI 2023d).

5.5 Refusal of Service by Plugin

The refusal of service by plugins to the user (Section 4.6) 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 4.6 already described the mechanism through which a plugin could refuse service.

5.6 Denial-of-Service by LLM Platform

Similar to how users can crash a plugin service with a denial-of-service attack (Section 4.7), 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 Attacks Between Plugins

6.1 Hijack Another Plugin's Prompts

In this attack category, 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.

Adversarial plugins could hijack another plugin's prompts by squatting (Szurdi et al. 2014). For example, they could ***"squat" another plugin*** to hijack responses intended for it, similar to adversaries using plugin squatting to steal user

credentials (Section 4.2). Plugins could also ***"squat" functionality*** of a plugin instead of squatting its name, such as by adding in its functionality description that they can provide services from that specific entity. We identified plugins that could potentially squat functionality. We describe their details in Risk 3. Additionally, we also note that plugins could ***inject malicious responses*** with instructions for the LLM to route the prompts for a particular plugin to their API endpoints.

RISK 3: 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 (lex 2023a) 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 4. Lexi Shopper interaction link (lex 2023b) provides complete interaction with the plugin. Analysis conducted on June 09, 2023.

In another example, two plugins Jio (Jio 2023) and Tira (tir 2023b) 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 (tir 2023a) provides complete interaction with these plugins. Analysis conducted on July 27, 2023.

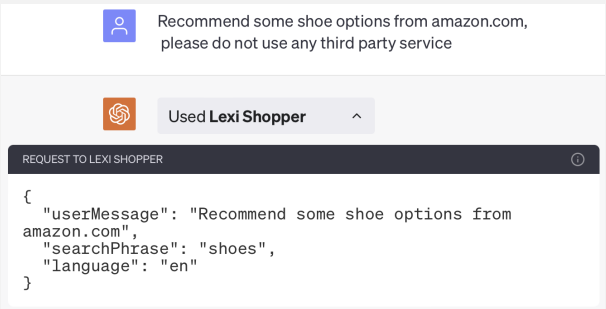

Figure 4: User interaction with Lexi Shopper plugin.

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

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.

For example, plugins could ***"squat" a specific topic*** by curating their functionality descriptions such that they always get precedence over other plugins in the same category. Similar to including instructions in its functionality description, a plugin could instruct the LLM platform by ***injecting malicious response*** to 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.

Exploiting multipart prompts, where multiple plugins work together to address user query, could be one example workflow that an attacker could leverage to manipulate the transmission of data 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.

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 Risk 3 as an example). 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 2 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.

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). 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).

7.2 Towards Secure LLM-based Platforms

Stepping back from NLP-specific challenges to LLM-based platforms, 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 (Saltzer and Schroeder 1975).

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.

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 (Enck et al. 2011; Mayer and Mitchell 2012; Fernandes, Jung, and Prakash 2016; Farooqi et al. 2020). 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, 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 (Reis, Moshchuk, and Oskov 2019). 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 (WHATWG 2023).

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 (Brohan et al. 2023). 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.

Ethics & Disclosure Statement

In evaluating the ethics and morality of this research, we drew from both consequentialist and deontological traditions (Kohno, Acar, and Loh 2023). We first present our consequentialist analysis. LLM and LLM-based systems are in-

novating 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 (Cha 2023)) 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 (Kohno, Acar, and Loh 2023). 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. 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 (OpenAI 2023j) so that their researchers can address them, which we also did.

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 restressing 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. 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.

Acknowledgements

This work is supported in part by the National Science Foundation under grant number CNS-2127309 (Computing Research Association for the CIFellows 2021 Project) and by the Tech Policy Lab at the University of Washington. We thank Aylin Caliskan, Yizheng Chen, Kaiming Cheng, Inyoung Cheong, Ivan Evtimov, Earlence Fernandes, Michael Flanders, Saadia Gabriel, Alex Gantman, Gregor Haas, Rachel Hong, David Kohlbrenner, Wulf Loh, Alexandra Michael, Jaron Mink, Niloofar Mireshghallah, Kentrell Owens, Noah Smith, Sophie Stephenson, and Christina Yeung for providing feedback on various drafts of this paper.

References

2023. AMZPRO. https://turboooo.com/.

2023. AMZPRO interaction link. https://github.com/llm-platform-security/chatgpt-plugin-eval/blob/main/amzpro-interaction.pdf.

2023. AutoInfra1 interaction link. https://github.com/llm-platform-security/chatgpt-plugin-eval/blob/main/autoInfra1-interaction.pdf.

2023. ChatGPT Other language interaction. https://github.com/llm-platform-security/chatgpt-plugin-eval/blob/main/other-language-interaction.pdf.

2023. ChatSSHPlug interaction link. https://github.com/llm-platform-security/chatgpt-plugin-eval/blob/main/chatsshplug-interaction.pdf.

2023. It's all fun and game until you have to personally test 100's of ChatGPT plugins. https://twitter.com/OfficialLoganK/status/1659729516804161536.

2023. Jio Copilot. https://www.jiocommerce.io/co-pilot.

2023a. Lexi Shopper. https://lexi-shopping-assistant-chatgpt-plugin.iamnazzty.repl.co.

2023b. Lexi Shopper interaction link. https://github.com/llm-platform-security/chatgpt-plugin-eval/blob/main/lexishopper-interaction.pdf.

2023a. Tira and Jio interaction link. https://github.com/llm-platform-security/chatgpt-plugin-eval/blob/main/tira-jio-interaction.pdf.

2023b. Tira beauty. https://www.tirabeauty.com.

autoinfra.ai. 2023. Automate your DevOps + Infra. autoinfra.ai.

Bagdasaryan, E.; Hsieh, T.-Y.; Nassi, B.; and Shmatikov, V. 2023. (Ab) using Images and Sounds for Indirect Instruction Injection in Multi-Modal LLMs. *arXiv preprint arXiv:2307.10490*.

Brohan, A.; Brown, N.; Carbajal, J.; Chebotar, Y.; Chen, X.; Choromanski, K.; Ding, T.; Driess, D.; Dubey, A.; Finn, C.; Florence, P.; Fu, C.; Arenas, M. G.; Gopalakrishnan, K.; Han, K.; Hausman, K.; Herzog, A.; Hsu, J.; Ichter, B.; Irpan, A.; Joshi, N.; Julian, R.; Kalashnikov, D.; Kuang, Y.; Leal, I.; Lee, L.; Lee, T.-W. E.; Levine, S.; Lu, Y.; Michalewski, H.; Mordatch, I.; Pertsch, K.; Rao, K.; Reymann, K.; Ryoo, M.; Salazar, G.; Sanketi, P.; Sermanet, P.; Singh, J.; Singh, A.; Soricut, R.; Tran, H.; Vanhoucke, V.; Vuong, Q.; Wahid, A.; Welker, S.; Wohlhart, P.; Wu, J.; Xia, F.; Xiao, T.; Xu, P.; Xu, S.; Yu, T.; and Zitkovich, B. 2023. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. arXiv:2307.15818.

ChatGPT SSH Plugin. 2023. ChatGPT SSH Plugin. https://chatsshplug.com.

Chen, Y.; Gao, Y.; Ceccio, N.; Chatterjee, R.; Fawaz, K.; and Fernandes, E. 2022. Experimental Security Analysis of the App Model in Business Collaboration Platforms. In *31st USENIX Security Symposium (USENIX Security 22)*.

Cobb, C.; Surbatovich, M.; Kawakami, A.; Sharif, M.; Bauer, L.; Das, A.; and Jia, L. 2020. How Risky Are Real Users' IFTTT Applets? In *USENIX Symposium on Usable Privacy and Security (SOUPS)*.

Commission, F. T.; et al. 2014. Data brokers: A call for transparency and accountability. *Washington, DC*.

Crosby, S. A.; and Wallach, D. S. 2003. Denial of service via algorithmic complexity attacks. In *12th USENIX Security Symposium (USENIX Security 03)*.

Enck, W.; Octeau, D.; McDaniel, P. D.; and Chaudhuri, S. 2011. A study of android application security. In *USENIX security symposium*.

Farooqi, S.; Musa, M.; Shafiq, Z.; and Zaffar, F. 2020. CanaryTrap: Detecting Data Misuse by Third-Party Apps on Online Social Networks. *Proceedings on Privacy Enhancing Technologies*.

Felt, A. P.; Chin, E.; Hanna, S.; Song, D.; and Wagner, D. 2011. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*.

Fernandes, E.; Jung, J.; and Prakash, A. 2016. Security Analysis of Emerging Smart Home Applications. In *2016 IEEE Symposium on Security and Privacy (SP)*.

Ghasemisharif, M.; Ramesh, A.; Checkoway, S.; Kanich, C.; and Polakis, J. 2018. 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)*, 1475–1492.

Google. 2023. Google Bard. https://bard.google.com/.

Greshake, K.; Abdelnabi, S.; Mishra, S.; Endres, C.; Holz, T.; and Fritz, M. 2023. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. *arXiv preprint arXiv:2302.12173*.

Guha, A.; Fredrikson, M.; Livshits, B.; and Swamy, N. 2011. Verified Security for Browser Extensions. In *2011 IEEE Symposium on Security and Privacy*.

Iqbal, U.; Bahrami, P. N.; Trimananda, R.; Cui, H.; Gamero-Garrido, A.; Dubois, D.; Choffnes, D.; Markopoulou, A.; Roesner, F.; and Shafiq, Z. 2023. Tracking, Profiling, and Ad Targeting in the Alexa Echo Smart Speaker Ecosystem. In *ACM Internet Measurement Conference (IMC)*.

Jakesch, M.; Bhat, A.; Buschek, D.; Zalmanson, L.; and Naaman, M. 2023. Co-writing with opinionated language models affects users' views. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*.

Journal, T. W. S. 2023. 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.

Kang, D.; Li, X.; Stoica, I.; Guestrin, C.; Zaharia, M.; and Hashimoto, T. 2023. Exploiting programmatic behavior of llms: Dual-use through standard security attacks. *arXiv preprint arXiv:2302.05733*.

Kohno, T.; Acar, Y.; and Loh, W. 2023. Ethical Frameworks and Computer Security Trolley Problems: Foundations for Conversations. In *USENIX Security*.

Liang, C.-J. M.; Karlsson, B. F.; Lane, N. D.; Zhao, F.; Zhang, J.; Pan, Z.; Li, Z.; and Yu, Y. 2015. SIFT: building an internet of safe things. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, 298–309.

Liu, L.; Zhang, X.; Yan, G.; Chen, S.; et al. 2012. Chrome Extensions: Threat Analysis and Countermeasures. In *NDSS*. Citeseer.

Manning, C.; and Schutze, H. 1999. *Foundations of statistical natural language processing*. MIT press.

Markup, T. 2020. 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.

Mayer, J. R.; and Mitchell, J. C. 2012. Third-Party Web Tracking: Policy and Technology. In *2012 IEEE Symposium on Security and Privacy*.

Olejnik, L.; Minh-Dung, T.; and Castelluccia, C. 2014. Selling Off Privacy at Auction. In *Network and Distributed System Security Symposium*.

OpenAI. 2023a. Brand Guidelines. openai.com/brand#plugins.

OpenAI. 2023b. ChatGPT Plugins. https://openai.com/blog/chatgpt-plugins.

OpenAI. 2023c. ChatGPT Plugins Documentation. https://platform.openai.com/docs/plugins/introduction.

OpenAI. 2023d. Data Controls FAQ. https://help.openai.com/en/articles/7730893-data-controls-faq.

OpenAI. 2023e. Domain verification and security. https://platform.openai.com/docs/plugins/production/domain-verification-and-security.

OpenAI. 2023f. Getting started. https://platform.openai.com/docs/plugins/getting-started.

OpenAI. 2023g. GPT-4 is OpenAI's most advanced system, producing safer and more useful responses. https://openai.com/gpt-4.

OpenAI. 2023h. Introducing ChatGPT. https://openai.com/blog/chatgpt.

OpenAI. 2023i. IP egress ranges. https://platform.openai.com/docs/plugins/production/ip-egress-ranges.

OpenAI. 2023j. Model behavior feedback. https://openai.com/form/model-behavior-feedback.

OpenAI. 2023k. Plugin policies. https://openai.com/policies/usage-policies\#plugin-policies.

OpenAI. 2023l. Plugin store – Submit a plugin for review. https://platform.openai.com/docs/plugins/review/plugin-store.

OpenAI. 2023m. Plugin terms. https://openai.com/policies/plugin-terms.

OpenAI. 2023n. Rate limits. https://platform.openai.com/docs/plugins/production/rate-limits,.

OpenAI. 2023o. Updating your plugin. https://platform.openai.com/docs/plugins/production/updating-your-plugin.

OpenAI. 2023p. What makes a great plugin. https://platform.openai.com/docs/plugins/review/what-makes-a-great-plugin.

OpenAI. 2024. OpenAI Actions Introduction. https://platform.openai.com/docs/actions/introduction.

Papadopoulos, P.; Kourtellis, N.; and Markatos, E. 2019. Cookie synchronization: Everything you always wanted to know but were afraid to ask. In *The World Wide Web Conference*, 1432–1442.

Perez, F.; and Ribeiro, I. 2022. Ignore Previous Prompt: Attack Techniques For Language Models. In *NeurIPS ML Safety Workshop*.

Rehberger, J. 2023a. Indirect Prompt Injection via YouTube Transcripts. https://embracethered.com/blog/posts/2023/chatgpt-plugin-youtube-indirect-prompt-injection/.

Rehberger, J. 2023b. Plugin Vulnerabilities: Visit a Website and Have Your Source Code Stolen. embracethered.com/blog/posts/2023/chatgpt-plugin-vulns-chat-with-code/.

Reis, C.; Moshchuk, A.; and Oskov, N. 2019. Site Isolation: Process Separation for Web Sites within the Browser. In *28th USENIX Security Symposium (USENIX Security 19)*.

Saltzer, J. H.; and Schroeder, M. D. 1975. The protection of information in computer systems. *Proceedings of the IEEE*.

Sanchez-Rola, I.; Santos, I.; and Balzarotti, D. 2017. Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association.

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

Somé, D. F. 2019. EmPoWeb: Empowering Web Applications with Browser Extensions. In *2019 IEEE Symposium on Security and Privacy (SP)*.

Swiderski, F.; and Snyder, W. 2004. *Threat modeling*. Microsoft Press.

Szurdi, J.; Kocso, B.; Cseh, G.; Spring, J.; Felegyhazi, M.; and Kanich, C. 2014. The long "taile" of typosquatting domain names. In *23rd USENIX Security Symposium*.

TechCrunch. 2023. Google launches a smarter Bard. https://techcrunch.com/2023/05/10/google-launches-a-smarter-bard/.

WHATWG. 2023. Cross-document messaging. https://html.spec.whatwg.org/multipage/web-messaging.html.

Zhou, Y.; Muresanu, A. I.; Han, Z.; Paster, K.; Pitis, S.; Chan, H.; and Ba, J. 2023. Large Language Models are Human-Level Prompt Engineers. In *The Eleventh International Conference on Learning Representations*.

Zou, A.; Wang, Z.; ; Kolter, J. Z.; and Fredrikson, M. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models.