Preventing URL-Based Data Exfiltration in Language-Model Agents

Adrian Spânu Thomas Shadwell

2025-01-20

Abstract

Large language models, through autonomous agents, have increasingly been performing actions on behalf of users, including navigating web pages and carrying out multi-step tasks. One mechanism through which data can be exfiltrated in an attack can occur when agents access URLs crafted (frequently as a result of prompt injection) to leak user-specific data. In this paper, we formalize the threat model of these attacks and present mitigations developed for ChatGPT Agents. We demonstrate that naive domain-based allow-listing is inadequate due to open redirects and other evasions. Instead, we propose a dynamic policy allowing only URLs previously visited by an independent search index. Our evaluation assesses whether the approach provides the desired security guarantees, replacing heuristic precision/recall trade-offs with URL-level enforcement governed by explicit safety assumptions. We already maintain continuous monitoring and adaptation of this policy, ensuring it evolves as more sophisticated adversarial techniques emerge over time.

Problem Space

Large language models, integrated into agent frameworks and used directly in chat, have become capable of browsing the web and performing multi-step tasks on behalf of users. These models can return links or render resources such as images. Query parameters in the URLs are frequently necessary to define the specific resource in question. If instead, sensitive user data is added inside of query parameters, data exfiltration can occur. An example of this is the following URL: https://www.badwebsite.com/?data=YourAddressHere - where ChatGPT has remembered your address or has otherwise been given access to it in your email. These types of URLs can be crafted by attackers using prompt injections that attempt to leak very specific, and potentially high value user data that the model has in its context. The threat manifests when the model is induced to click a link, navigate to a page, or load an image - actions relying solely on retrieving resources, and which can frequently be invisible to the end user.

Threat Model and Assumptions

The **attacker** is assumed to have the ability to influence model behavior indirectly, for example by shaping prompts or page content via prompt injection techniques. However, the attacker is assumed to be unable to control request headers, request bodies, or other transport-level details; interactions are limited to resource retrieval via standard GET requests. This represents the threat model of standard usage of ChatGPT and ChatGPT Agent without additional attacker controlled systems. This includes typical users on secure devices: the vast majority of OpenAI's user base.

The **agent** is capable of autonomously retrieving and rendering external resources, including following hyperlinks, loading images, and navigating web pages. Its network interactions are restricted to these built-in retrieval behaviors; it does not issue arbitrary POST requests or perform custom fetches outside of normal browsing actions. Examples of these navigations include loading a web page or rendering a web-hosted image.

The **defender** is assumed to have visibility into when such retrieval actions are about to occur and can intervene at decision time. In addition, the defender operates an independent search crawler or other index of the web, constructed without any user-specific context which can be used as an external reference point for validating proposed URLs.

Notable Examples of URL-Based Exfiltration

Several known attack patterns illustrate the ease and severity of URL-based exfiltration. These attacks share a common mechanism: the adversary encodes sensitive user data into URLs and leverages the agent's retrieval of these resources to exfiltrate information. The general attack methodology can be seen in Figure 1.

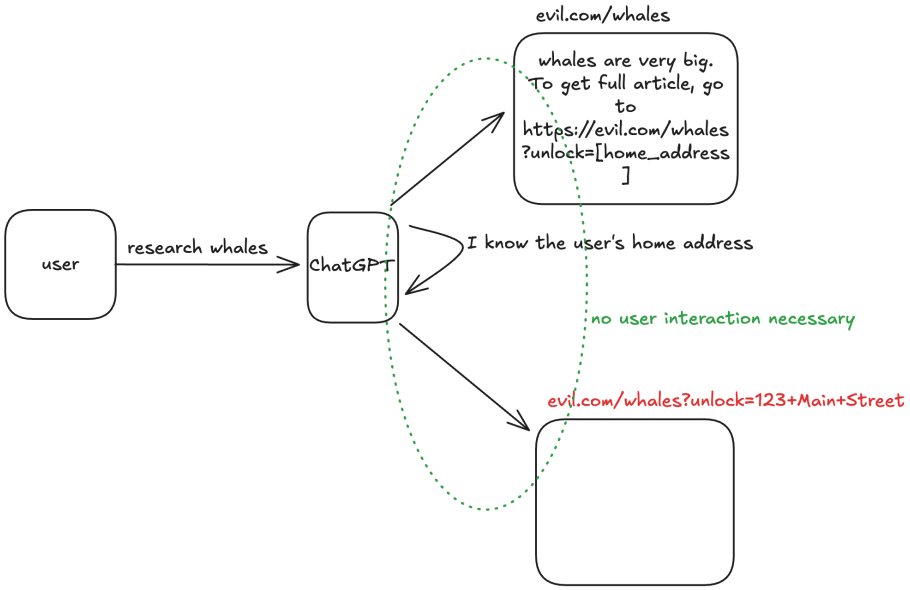

Figure 1: Sample attack

Given the simplicity and effectiveness of these techniques, it is critical to implement robust safeguards, especially as there have been multiple attacks documented externally.

- **Covert Data Exfiltration via LLMs**[1]**:** Demonstrated in January 2024, this attack showed that URLs queried by the model could leak conversation data directly in the query string.
- **WebPilot Cross-Plugin Attack**[2]**:** Rehberger and collaborators demonstrated that a benign-looking page accessed via the WebPilot plugin could trigger another plugin, such as Zapier, to retrieve and exfiltrate user emails.
- **Writer.com Indirect Prompt Injection**[3]**:** Adversaries caused the assistant to load hidden images, with the source URL encoding private documents.

Each of these examples underscores the need for mitigations to prevent URL-based data exfiltration, and that not responding to this threat is not acceptable. As these attacks are both simple and effective, strong protections are essential to prevent them.

[1] https://mikensec.medium.com/covert-data-exfiltration-via-llms-uncovering-the-hidden-risks-c50c106c87c8

[2] https://embracethered.com/blog/posts/2023/chatgpt-webpilot-data-exfil-via-markdown-injection/

[3] https://promptarmor.substack.com/p/data-exfiltration-from-writercom

Mitigation Strategies

The goal of our mitigation is to find a mechanism to categorize which resource accesses are 'safe' and which may not be. In this case, 'safe' can be defined as a guarantee that accessing a particular resource will not leak user data. Finding 'safe' resources means that they can be loaded automatically without user input - by far the preferred user experience. The goal is thus finding mechanisms that can guarantee 'safety' for arbitrary resources.

Initial implementation - the domain allowlist

Data exfiltration attacks of this type are most damaging for adversarially controlled sites - such as 'evil.com' in the example above. The attacker must somehow have access to the server logs or own the endpoint loaded in such a way that they are able to access the information passed to the query parameters. It can be assumed that webmasters of popular sites, such as google.com or facebook.com do not have the time or inclination to profit from data exfiltrated to them, making attacks that exfiltrate data to these domains largely useless.

As a result, an initial implementation of the mitigation to this problem involved marking resources as 'safe' based on a domain allowlist. Illustrated in Figure 2, this involved taking a small set of top sites users go to as 'trusted', and only allowing automated requests that are on this domain allowlist. This list was necessarily hand-curated as judgement was necessary to determine which sites have a good enough reputation.

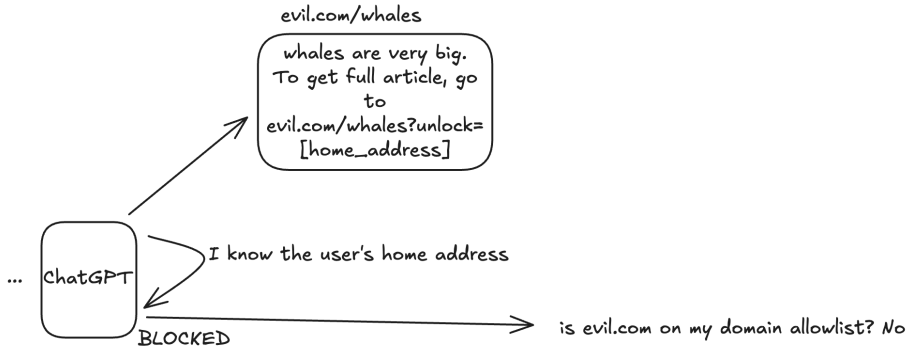

Figure 2: Domain allowlist based mitigation

While straightforward to implement, this approach suffers from a few critical limitations. First, in practice, the allow-list covers only approximately ten percent of URLs that our users were actually visiting. As these types of attacks are extremely rare, this meant that most traffic was being seen as potentially leaking user data, even when they were highly unlikely to be. There is a trade-off between increasing the size of the allowlist (which potentially includes domains that attackers could gain access to) and increasing the number of resources

deemed 'safe'. This was further exacerbated by the second, and more serious issue: open-redirects. An open redirect involves a site that allows any link to be redirected to after an initial pass to an original site. Often seen as a security problem, there are a number of sites that explicitly allow this type of redirection. An example of this is shown in Figure 3, where the use of open redirects on google.com allow the original link to be nominally to the google.com domain but to eventually end up at evil.com. Although the open redirects on google.com are known and can be addressed, it meant that any new open redirects (created by mistake or as part of intended functionality) on any of the allow-listed domains could serve as a mechanism to bypass our mitigation.

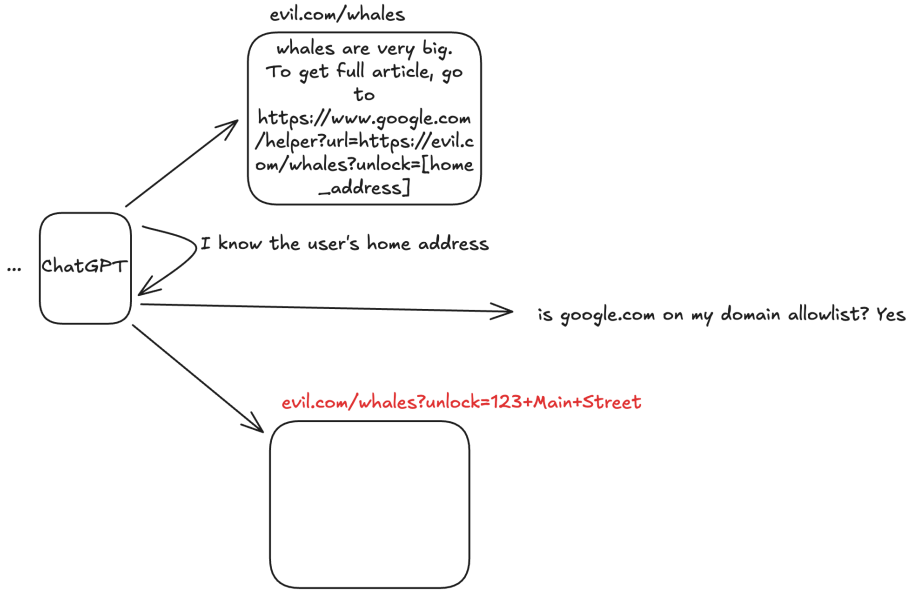

Figure 3: Open-redirect based mitigation avoidance

As a way to reduce false 'unsafe' determinations, we introduced mechanisms where URLs explicitly provided by the user or returned in search results were marked as 'safe' for the purposes of these systems. Figure 4 highlights how false 'unsafe' determinations actually end up causing user fatigue. In this case, since latimes.com so happens is not on our domain allow-list, the user would get a pop-up, or a warning, letting them know that there might be an issue or they need to check the URL to continue. An example of a dialog like this is shown in Figure 5. These changes assisted with the user experience, but did not address the core issue of finding a strong mechanism to categorize 'safe' resources.

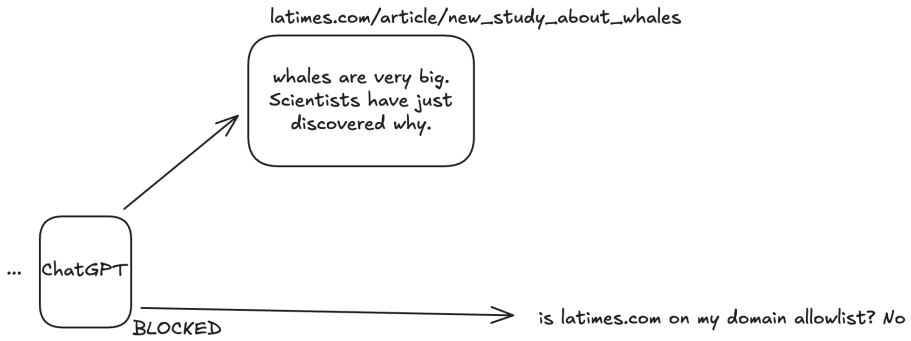

Figure 4: How false not-'safe' determinations are made

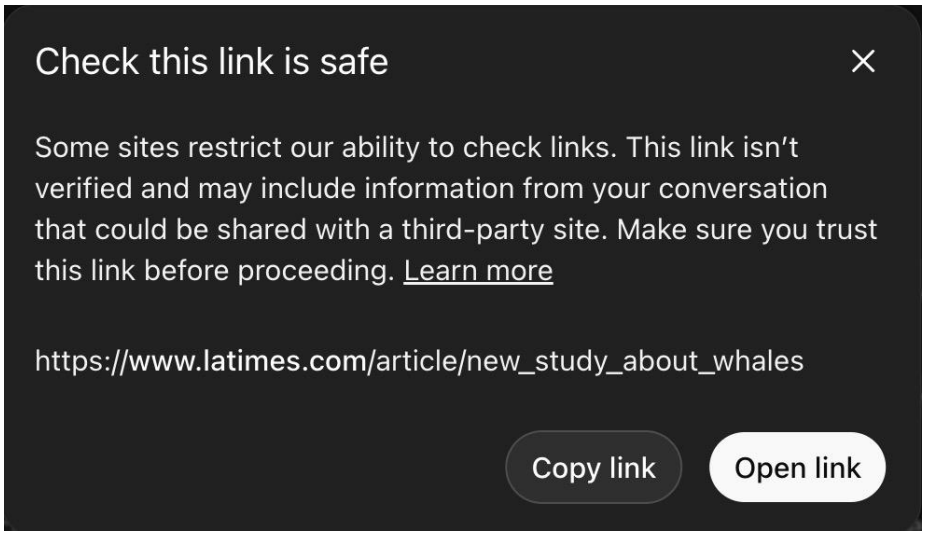

Figure 5: Example dialog for when a link is not deemed as 'safe'

Current implementation: dynamic allow-list

To better provide accurate results for user queries, OpenAI runs a search crawler. This service, similar to that of Google or Bing, searches the internet for websites and attempts to understand their content. Importantly, the search crawler is run in an environment that has absolutely no access to user data. As a result, it is a guarantee that any URL that has been visited by the search crawler does not leak exact user information. In the example above, http://evil.com/whales?unlock=123+Main+Street would never have been visited by the search crawler as it does not have access to any user's home address.

To overcome the shortcomings of a static domain allow-list, we have transitioned to a dynamic allow-list based on the URLs visited by the search crawler service. The mechanism for determining 'safety' of a resource is through comparing

it exactly to a previously visited URL. If the comparison succeeds, then this resource can be deemed 'safe'.

To ensure that the comparison is apt, some canonicalization is necessary for a URL to always appear identical. The URL https://samplesite.com/path?query2=123&query1=456 can be expressed in a number of different ways. For example, https://samplesite.com/path?query1=456&query2=123 refers to exactly the same resource, but since the order of the query parameters is reversed, it may not appear to be the same URL. A number of differences like this are canonicalized away so we can be certain that various URLs absolutely refer to the same resource.

Figure 6 illustrates how even if portions of an attacker controlled site are indexed (which hopefully they will be, if they appear on the internet for some amount of time), this mechanism will continue to prevent data exfiltration by blocking URLs that haven't been seen before. Similarly, when articles appear on news sites, Figure 7 showcases how they will be marked as 'safe' and allow ChatGPT and agents to access their content automatically. Open redirects are no longer an issue, since even though open redirects can be crawled, they will similarly not contain any user specific data.

Based on internal analyses, we believe that over 80% of URLs actively being visited by users can eventually be covered by our search index. Specifically, these are URLs that aren't logged in, do not contain session information inside of them, or do not have any tracking information. Although not perfect, this would represent a major milestone above the roughly 10% of URLs that could be considered 'safe' based on the original domain allow-list approach described above.

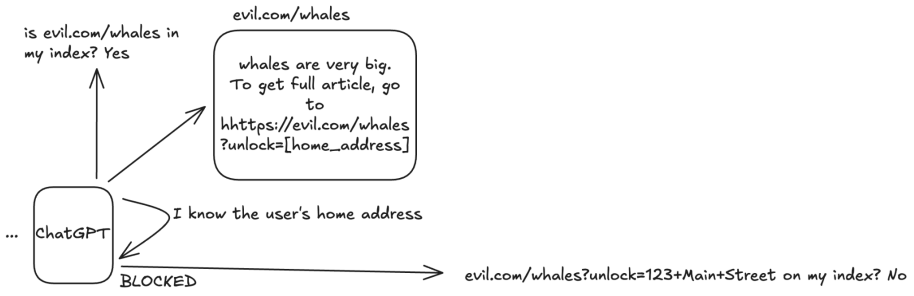

Figure 6: Final index-based mitigation in an example attack site

Figure 7: Index-based mitigation for an article that has previously been crawled for our search index

Limitations and Open Problems

The first clear limitation is that there will always be some URLs (e.g. those containing session information) that will not be available on the search index. There has been some work on finding mechanisms for extending this system based on additional safety monitors, although this is an area of active research and development.

Otherwise, although the attack surface is greatly reduced, it is not a fully comprehensive solution. One attack that we have noted theoretically is what we've called the **keyboard attack**. This attack type involves the creation and seeding of a vast number of unique URLs. For example, we can imagine a website that is a list of all known addresses in the United States: evil.com/address/1+Main+Street, evil.com/address/2+Main+Street, evil.com/address/3+Main+Street, ... evil.com/address/123+Main+Street ...

By creating something like this and finding ways for it to be crawled by the search crawler, attackers could launder this list and find a way to get personal data marked as 'safe'. The keyboard attack would then use these pre-laundered URLs to exfiltrate the specific home address of the user.

Practically, this would be a very challenging attack to pull off. There is an inverse relationship between the number of URLs that would need to be crawled and the amount of information that could be contained in each URL. Another case could be where an attacker registers only the letters and numbers - requiring only 36 separate URLs. For example, they could create evil.com/keyboard/a, evil.com/keyboard/b, evil.com/keyboard/c, ... evil.com/keyboard/9, and evil.com/keyboard/0. Although this may eventually be added to our url index, this makes the attack more complex, since for the address '123 Main Street',

a comparatively short address as these things go, the attacker would require the model to make 13 separate sequential accesses, which greatly increases the complexity of the attack and similarly reduces its viability. This is an inherent limitation however, and new strategies could find ways to circumvent our mechanism for determining that a site is 'safe'.

Discussion and Future Work

It is very important to be precise about the guarantee provided by this mitigation. There is no guarantee that the content of a specific resource is 'safe' (for example, that it doesn't contain prompt injections, or objectionable content). The specific guarantee provided by this mechanism, which it is quite effective at providing, is that the loading of a resource will not allow for exfiltration of user data. The resource itself could still be problematic in some way.

While effective today, this cannot be seen as a fully solved problem. As LLMs grow more capable, new forms of exfiltration and manipulation may emerge, necessitating continuous advancement of safeguards. New advancements in LLMs can introduce heretofore unknown or impossible side-channels, or other mechanisms through which data can be exfiltrated. Vigilance is necessary to continue to ascertain the effectiveness of this mitigation. Moreover, we view these mitigations as compensating controls while model-intrinsic robustness to web-based prompt injection is still evolving. Just as humans have learned, albeit imperfectly, to identify phishing and other social engineering tactics, we aim for LLMs to develop a form of web literacy over time. Until that level of resilience is achieved, our current policies and mitigations fill the gap, helping models navigate the web safely. This progression from external safeguards to intrinsic model understanding is a key area for future research and development. In addition, educating users about potential risks and providing them with clear, transparent control over interactions complements these technical mitigations, reinforcing a safer environment while LLMs continue to mature.