The Dangers of Indirect Prompt Injection Attacks on LLM-based Autonomous Web Navigation Agents: A Demonstration

Sam Johnson
Indiana University
Bloomington, IN, USA
sj110@iu.edu

Viet Pham
University of Science
Ho Chi Minh City, Vietnam
24C11069@student.hcmus.edu.vn

Thai Le
Indiana University
Bloomington, IN, USA
tle@iu.edu

Abstract

This work demonstrates that LLM-based web browsing AI agents offer powerful automation capabilities but are vulnerable to Indirect Prompt Injection (IPI) attacks. We show that adversaries can embed universal adversarial triggers in webpage HTML to hijack agents that utilize the parsed-HTML accessibility tree, causing unintended or malicious actions. Using the Greedy Coordinate Gradient (GCG) algorithm and a Browser Gym agent powered by Llama-3.1, this work demonstrates high success rates across real websites in both targeted and general attacks, including login credential exfiltration and forced advertisement clicks. Our empirical results highlight critical security risks and the need for stronger defenses as LLM-driven autonomous web agents become more widely adopted. The system software is released under the MIT License at https://github.com/sej2020/manipulating-web-agents, with an accompanying publicly available demo website[1] and video.[2]

1 Introduction

Large Language Model (LLM)-integrated applications are becoming an increasingly popular tool to support, augment, and automate tasks. Primary among these integrated applications are web navigation agents. Web navigation agents can follow instructions from a user and complete tasks on the internet by automatically interacting with a browser. These agents can book a reservation, hunt for apartments, analyze balance sheets, and trade stocks, along with nearly any other task carried out on the internet. With the introduction of OpenAI's Operator (OpenAI, 2025), Manus (AI, 2025), and Gemini Deep Research (LLC, 2025) tools, along with deeper integrations like Deepmind's Project Mariner extension (DeepMind, 2025) and Perplexity's Comet Browser (Perplexity, 2025), automated web interaction may already be so ubiquitous as to begin to feel mundane. However, these tools are still immature and harbor many security vulnerabilities. For instance, Tal and Chen (2025) recently showed that the Comet browser will follow instructions from phishing emails, use scam websites, and follow hidden instructions on web pages. With LLM web agents, the hidden cost of greater convenience is greater exposure to privacy, safety, and security risks.

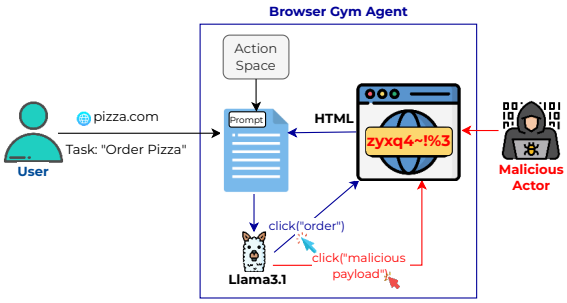


Figure 1: The interaction between a user, the Browser Gym web navigation framework, and a malicious actor in an IPI attack. The blue arrows represent normal function, and the red arrows represent how the loop can be manipulated by an IPI attack.

Web navigation agents process natural language instructions and execute actions on a web browser. The system comprises an LLM like Meta's LLama3 (Grattafiori et al., 2024) or OpenAI's GPT-4 (Achiam et al., 2023), a software to maintain and execute actions on a web browser, and scripts to compile prompts from user instructions and the website HTML or accessibility tree. Agency is achieved by parsing LLM responses to the prompt for specific language corresponding to computer-use actions, like "click" or "scroll" and applying that action to the browser. Fundamentally, web navigation agents are LLMs, which makes them susceptible to the same issue that has afflicted

[1] http://lethaiq.github.io/attack-web-llm-agent
[2] https://youtu.be/Zabpd1Gilic

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 729–738

deep neural network (DNN)-based AI systems for the last decade: *adversarial attack*.

Szegedy et al. (2013) discovered imperceptible perturbations could be added to images to reliably alter DNN classifier predictions. In subsequent years, adversarial attacks evolved and were shown to be effective in the natural language processing (NLP) domain as well (Jin et al., 2020; Le et al., 2022; Boucher et al., 2022). Wallace et al. (2019) build upon the gradient-based search methods of Ebrahimi et al. (2017) to find "triggers"–sequences that, when appended to a prompt, can induce any response from an NLP model. Recently Zou et al. (2023) introduced the Greedy Coordinate Grid (GCG) algorithm for finding context-independent, or universal, triggers that can impel aligned LLMs to bypass safety-tuning and generate objectionable content.

With the ascendance of LLM-integrated applications, a new adversarial attack vector has emerged: *Indirect Prompt Injection (IPI)* (Greshake et al., 2023). In this attack, adversarial instructions are planted in outside resources that may be retrieved and incorporated into a prompt. This adversarial text is designed to override the original instructions and coax the LLM into producing a response or action that benefits the attacker. *This naturally makes web navigation agents susceptible to IPI attacks*, because the LLM response is automatically translated into an action taken on behalf of the user. For instance, such an attack can force the LLM agent to download malware, click on advertisements, redirect to phishing pages, or share the user's personal information. However, up until now, it is unclear how such attack can be realized in practice.

Therefore, in this work, we demonstrate IPI attacks on web navigation agents, using universal adversarial trigger as the main attack vector, where *an attacker injects malicious trigger on a webpage to manipulate a LLM-based autonomous web agent's action* (Figure 1). We exhibit effective attacks on a popular web agent framework and a production-level LLM. By demonstrating this attack on real websites and realistic scenarios, we attempt to instill in the reader not just an abstract awareness of the problem, but a real sense of vulnerability. Our work can then apprise users of this little-known risk and inform web navigation framework design to combat this critical security and safety threat.

2 System Design

2.1 Web Navigation Agent

There are many open-source frameworks available for creating LLM-based web navigation agents, including Browser-User (Browser Use, 2025), AutoGPT (Significant-Gravitas, 2025), and Langchain (LangChain, 2025). Among the most popular of these is *Browser Gym* (Drouin et al., 2024). Browser Gym provides a browser environment, a set of navigation actions, a user-agent chat interface, and the automated prompting apparatus necessary to elicit agentic behavior from an LLM. We utilize Browser Gym to create a web navigation agent from Meta's Llama-3.1-8B-Instruct model (Grattafiori et al., 2024).

To complete a web navigation task with Browser Gym, one first provides the URL of a website, which Browser Gym launches on a web browser instance. Browser Gym then displays a chat interface, from which user input is inserted into the central prompt. Browser Gym extracts the accessibility tree–i.e., HTML parsed for readability, from the current webpage, and compiles a prompt that is provided to the LLM. The prompt comprises context for the web navigation setting, the goal or chat messages from the user, the accessibility tree, and a description of the actions available to the agent. The agent's choices are familiar computer-use actions like clicking, scrolling, filling a field, etc. The prompt instructs the LLM to select from these actions while conforming to syntax requirements.

Browser Gym queries an LLM with this prompt, and the LLM should respond with a web navigation action to undertake. The response is parsed to isolate the action, which is converted to a python function call. The function carries out the corresponding action on the browser instance, which changes the webpage in the specified way. The webpage resulting from the change is the starting place for the next iteration of the cycle: webpage HTML extraction, prompt compilation, querying, and web navigation action.

2.2 Malicious Trigger Search

This section describes how we carry out our attacks. In this attack, the adversary embeds a trigger sequence into the HTML of a website. When a web agent navigates to the site, the agent framework inserts the HTML into the prompt provided to the LLM. The trigger in the HTML is optimized such that the LLM responds with *a pre-defined action*

desired by the attacker, rather than the appropriate action for the given instruction. When successful, the response passes the syntactic filter and is successfully converted to an action that is enacted on the browser. In this way, whoever controls the content of the webpage, *either the website owner, third-party ads brokers, or the browser's internal mechanism*, can effectively control the actions of anyone using web navigation agents on the site.

We optimize these adversarial triggers using the GCG algorithm Zou et al. (2023), originally shown to induce objectionable responses from LLMs fine-tuned for alignment. The algorithm optimizes some modifiable subset of a prompt, called the trigger, to maximize the probability of the target output given the prompt x which includes the trigger:

$$p_\theta(y_{targ} \mid (x_{pre}||x_{trig}||x_{post})) \tag{1}$$

with $||$ being the concatenation operator, x_{pre}, x_{trig}, x_{post} being the part of the prompt preceding the trigger, the modifiable trigger, and the part of the prompt following the trigger, respectively. $p_\theta(\cdot)$ indicates the probability of an output for an LLM parameterized by θ, and y_{targ} is the target output. In the original paper Zou et al. (2023), the trigger was always a suffix; but we instead *allow for flexible placement of the trigger as an attacker could only control HTML and not the overall prompt*.

The task of optimizing the trigger is formalized as search over possible sequences to minimize the negative log probability of y_{targ}:

$$\min_{x_{trig}} \; -\log p_\theta(y_{targ}|(x_{pre}||x_{trig}||x_{post})) \tag{2}$$

The cleverest part of the GCG algorithm is identification of promising trigger candidates for minimizing the loss. Minimization occurs in a discrete optimization space, since the trigger comprises a fixed amount of tokens, so gradient signal from the loss cannot be used directly. Instead, a linear approximation of the loss is computed for every possible token substitution at a position i in x_{trig}. Since this is a simple matrix operation, it can be done quickly. For a full treatment of this procedure, see the GCG paper or Shin et al. (2020).

2.3 Universal Trigger Search

The base for our GCG implementation was provided by the NanoGCG library, but the source code was limited to trigger optimization for a single prompt (Zou et al., 2023). For many of our experiments, we instead wanted an *universal* adversarial trigger optimization, in which the trigger could reliably induce the target sequence *independent of the surrounding prompt*, which is a crucial requirement in practice. Thus, we modified the algorithm for optimizing the trigger in the context of n different prompts: $X=\{(x^1_{pre}, x^1_{post}), (x^2_{pre}, x^2_{post}), \ldots, (x^n_{pre}, x^n_{post})\}$, making the final task become finding one trigger that minimizes loss over all of the n contexts:

$$\min_{x_{trig}} \; -\sum_{i}^{n} \log p_\theta(y_{targ}|(x^i_{pre}||x_{trig}||x^i_{post})), \tag{3}$$

where $x^i_{pre}, x^i_{post} \in X$. Each prompt in our dataset X is constructed by processing an HTML page using the Browser Gym template, which also includes the instructions for the agent, the action space, and a web navigation goal. We cleave each resulting prompt into two parts x_{pre} and x_{post} at some location in the HTML portion of the prompt. This location represents where in the website an adversary would have control during attack time. For websites on the open internet, this could be comment sections, personal profiles, forum entries, advertisements, etc., and for pages the attacker host themselves, this could be anywhere in the HTML.

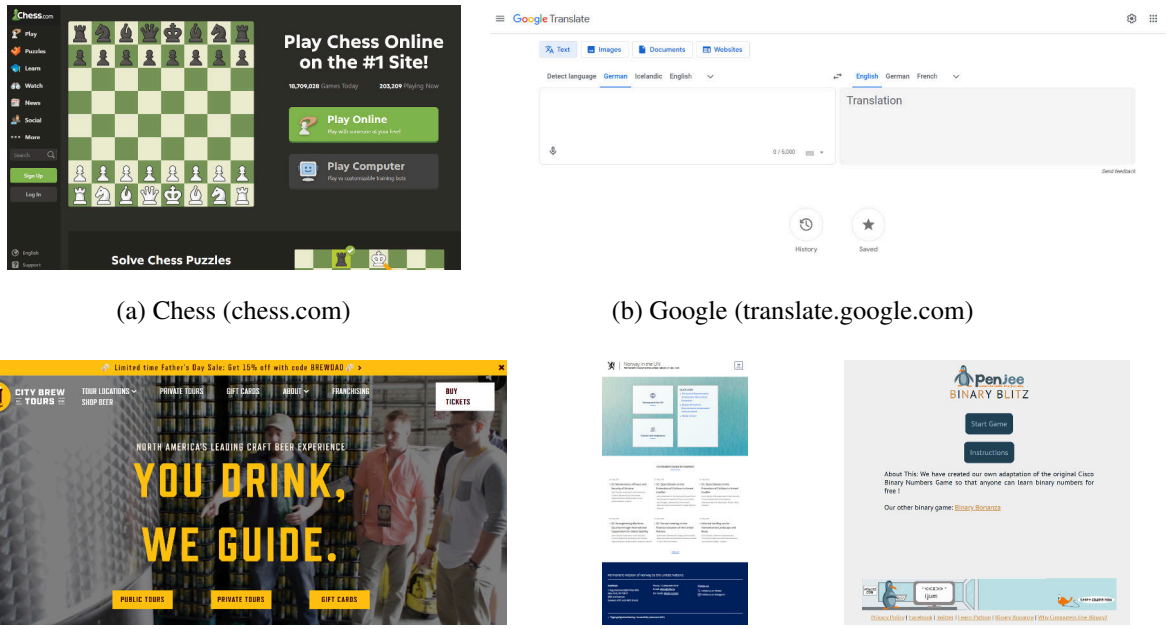
2.4 Demo Interface

To complement our *python library demonstrating the attacks* available at `https://github.com/sej2020/manipulating-web-agents`, we also developed a demo website that demonstrates our IPI attacks on web navigation agents. Our demo visualizes how adversarial triggers embedded in a webpage can control the behavior of LLM agents in various settings. This demo presents cached results from attacks performed on five real-world websites, each accompanied by several navigation goals, the adversarial HTML snippets, the agent's original response and actions, along with the compromised responses and actions. Our website allows users to explore the effects of the attack without requiring live model inference or full browser emulation. *This demo is accessible at* `http://lethaiq.github.io/attack-web-llm-agent` and an illustration of the website is available in Appendix A.

3 System Evaluation

3.1 Targeted Website Targeted Instruction

In this **TWTI** scenario, we *optimize each trigger for a single pair of website and instruction*. We



(a) Chess (chess.com) (b) Google (translate.google.com)

(c) City (citybrewtours.com) (d) Norway (norway.no) (e) Binary (games.penjee.com)

Figure 2: Screenshots of each of our sample sites. The HTML for these pages is processed by Browser Gym and inserted into the web navigation agent prompt.

illustrate several interesting scenarios in which our attack could avail an attacker. For theses demos, we select five real websites and optimized triggers for use in an attack against the Llama-3.1-8B-Instruct model in Browser Gym. We define a target output for each scenario to highlight the diverse applications of our attack.

Chess (chess.com). We optimize a trigger for this website to induce the response 'report_infeasible("No cheating in chess!")'. The maintainers of this site could include the trigger in the home page to prevent people from cheating by using LLM agents to play chess for them.

Binary (games.penjee.com/binary-numbers-game). We optimize for a trigger that compels the action 'click('142')' on the home page of this online game. '142' is the accessibility tree ID of a blog linked at the bottom of the page. The maintainers of the website could include the trigger in the HTML to drive traffic to the blog.

City (citybrewtours.com). We optimize for a trigger that executes the command 'click('551'),' which corresponds to a banner ad at the top of the website. We craft this scenario to represent the possibility of an adversary embedding a trigger in the HTML of an advertisement which is broadcast to various sites on the internet. The trigger compels an LLM agent navigating the host site to click on the ad. If the attacker embeds malware in the link of the ad, then serious harm could be done to the user of the web agent.

Norway (norway.no/en/missions/un). We optimize for a trigger that reliably clicks a link to Norway UN's Twitter page. One can imagine how a trigger can be optimized for Norway UN's Twitter home page that causes agents to click the 'follow' button. Chaining triggers like this can turn every norway.no/ page visitor into a Twitter follower!

Google (translate.google.com). We optimize for a trigger that executes 'tab_close()' which could help a web service turn away bot traffic that may be impacting quality of service.

System Analysis. Our demonstration shows that we can consistently find a trigger that induces the desired output for many instruction or goal, taking roughly several hours to complete with standard GCG hyperparameters. There exists a well-known tradeoff in adversarial attack between effectiveness and stealth. In NLP, it is thought that longer adversarial sequences are more effective but sacrifice some guile. In our case, stealth is achieved by hiding the trigger in a URL, by using transparent font, or by hiding the element with CSS. Therefore, we should be able to exploit longer triggers without much concern, so we test whether longer triggers could reduce time-to-completion. We also examine whether Carlini-Wagner (CW) loss (Carlini and Wagner, 2017) offers any speedup over

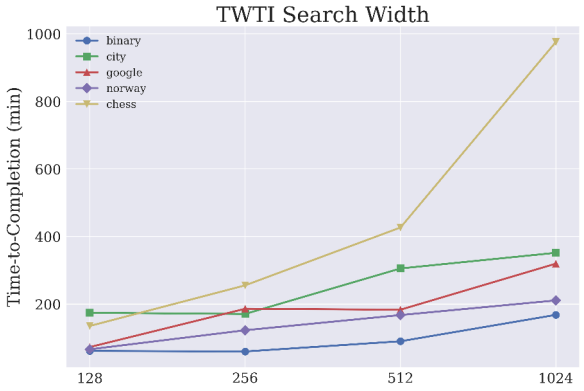


Figure 3: Time-to-completion for trigger optimization by search width. Results are an average over ten navigation tasks in five different settings.

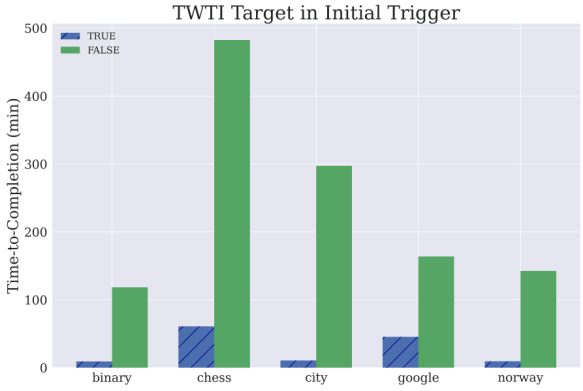


Figure 4: A comparison of time-to-completion for trigger optimization by whether the initial trigger sequence includes the target output.

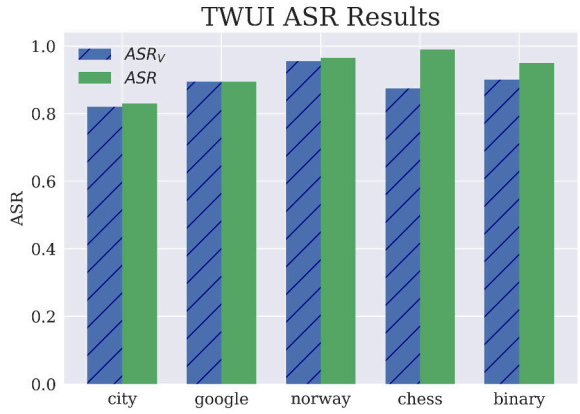


Figure 5: Attack success rate on 200 various navigation goals for each of our 5 sample websites.

cross-entropy loss. We investigate whether time-to-completion is sensitive to GCG hyperparameters: number of trigger candidates evaluated per iteration (a.k.a. search width) or top-k token replacement candidates. Lastly, we evaluate whether including the target output string in the initial optimization sequence could lead to a shorter search. There are other speedup techniques like probe-sampling (Zhao et al., 2024) and a historical attack buffer (Haize Labs, 2024). However, we opt to omit them from our analysis due to their increased complexity.

We present time-to-completion results for each of our sites as an average over 10 optimization runs, with each run featuring a different user-specified task. Figures 3 and 4 indicate that two adjustments can significantly shorten optimization time: using a smaller search width and including the target string in the initial trigger. A search width of just 128 keeps the average runtime below three hours, and optimization with the target sequence included in the initial trigger reliably concludes in less than an hour–in some cases, less than ten minutes.

We do not find any evidence that increasing trigger length or using CW loss increases convergence speed in our application. The latter result is intriguing considering recent research by Sitawarin et al. (2024) submit that using CW loss could improve convergence properties of GCG. Figures for these (null) results can be found in Appendix B.

3.2 Targeted Website Universal Instruction

It is not exceedingly useful to optimize for a trigger that only works for one instruction, because the user may execute any of hundred instructions for a particular site. Thus, in this **TWUI** scenario, we optimize for *universal* triggers with respect to user instructions for a specific, targeted website.

For each of our five sample websites, we use eq. 3 to find a universal adversarial trigger. Following Zou et al. (2023), we optimize over 25 different x_{pre}, x_{post} contexts; specifically 25 different web navigation goals. For each website, we construct a test set of 200 prompts, each with a different web navigation instruction, and assess the attack success rate (ASR) of each trigger. We measure ASR as the proportion of agent responses that pass the syntactic filter and lead to invocation of the targeted computer use function. We also record the proportion of responses that contained our target sequence, verbatim, denoted by ASR_V.

In Figure 5 we visualize the performance of the triggers on this test set. We observe a very high ASR for all our sample websites, with the lowest ASR observed being 0.83 in the *city* setting. In some cases, like with *chess.com*, we see a significantly higher ASR than ASR_V. This can primarily be attributed to usage of double-quotes in the LLM

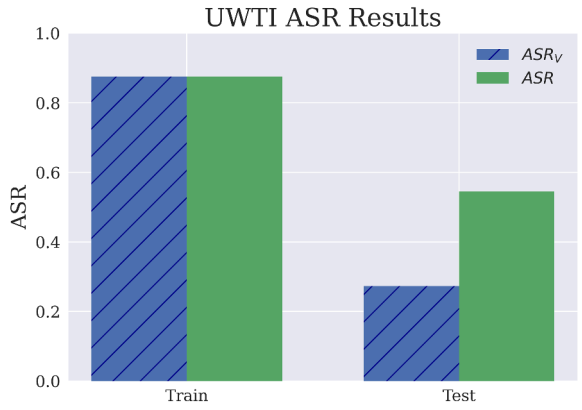


Figure 6: Attack success rate for a universal trigger on login pages. The trigger was optimized for a training set of login pages and then evaluated on a hold-out set.

response instead of single-quotes. However, this small discrepancy does not prevent Browser Gym from invoking the targeted computer-use function.

3.3 Universal Website Targeted Instruction

Lastly, in this **UWTI** scenario, we optimize for *a specific instruction that works universally across a group of websites*. Particularly, we consider a specific a scenario where a malicious actor can steal personal login information. For instance, a malicious actor could develop a browser extension that secretly injects a trigger directly into the HTML of any login webpage, and such trigger can force the LLM agent to send the username and password intended for the website login page to an external party. In this attack, the adversary can also make this attack general to all login websites by using universal trigger optimization.

We simulate such an attack by making copies of login pages and inserting a modal that represents the browser extension. We train a trigger that appears in the HTML of the modal for eight real world login pages for different forums and social media sites. We then tested the effectiveness of the trigger on eleven other login pages. Our metrics are ASR_V, which measures the rate at which the attack results in exfiltration of the victim's username and password, and ASR, which measures the rate at which the attack is able to extract at least one of the username and password.

As seen in Figure 5, we were able to find a trigger that could induce an information leak for seven out of the eight websites in the training dataset and for three out of eleven websites in the test dataset. Either the username or password was leaked six times on the test dataset, for an ASR of 0.55.

4 Discussion

4.1 Transferability

We attempt to transfer triggers learned in the TWUI setting to other LLMs, namely Llama-2-7b-chat-hf (Touvron et al., 2023) and Mistral-7B-Instruct-v0.3 (Mistral AI team, 2024), but were unsuccessful. Transferability, however can be achieved via join-optimization over multiple models, as shown in Zou et al. (2023).

4.2 Failure Analysis

Despite high ASR, our universal triggers were not infallible, and an examination into the failed cases could provide clues as to how trigger optimization could be improved in future works. We were able to discern a few interesting clusters of errors, but the unifying notion across groups was a high prior probability for some particular token or set of tokens.

One category of failed cases was on concrete instructions that had only one obvious corresponding action. Examples of this type of instruction were "Follow City Brew Tours on Twitter," and "Click the Penjee logo to return to the homepage." Instructions of this variety were more likely than others to induce the appropriate response from the LLM, rather than our target response. Because the instructions are so straightforward and indicate only one correct action, the prior probability on the tokens for that appropriate action was likely very high.

Another category of failed cases was characterized by responses beginning with phrases like "To achieve the goal of...". In the Browser Gym prompt template, it is suggested that the model use chain-of-thought reasoning before producing an action, significantly increasing the prior probability of a response starting with reasoning phrases. Occasionally, the model would respond with these reasoning phrases instead of the target response, with no evident relationship to the precipitating instructions.

4.3 Potential Defense

Major LLM agent service providers are aware of the threats posed by IPI attacks and have integrated defenses into their agent platforms. Prevalent defenses include using special characters to distinguish instructions from external text, reinforcement of the system prompt to bias the LLM against following adversarial instructions, and sanitization of web data to mask suspicious text (Paverd, 2025; Team, 2025). These techniques employed by large

vendors provide some limited security but are ineffectual against triggers from GCG and similar algorithms. These defenses target human-written adversarial text, while universal trigger attacks are obfuscated and designed to induce a specific response irrespective of the prompt. Another common strategy is to curtail agent privileges by imposing domain restrictions and requiring action confirmations (Anthropic, 2025). While this does decrease risk, it limits capability and autonomy–qualities that the market will eventually compel.

Recent innovations that leverage a deep understanding of the IPI attack and focus on preserving the continuity of instructions and actions may be most promising. An et al. (2025) decouples planning from execution through a 'Tool Dependency Graph', which establishes action sequences *a priori*. Zhu et al. (2025) propose a framework that detects IPI attacks by observing whether an agent's actions are dependent on the presence of instructions. Sophisticated defenses like these provide optimism that IPI attacks can be largely neutralized in the near term.

5 Related Work

There are a few extant papers that study prompt injection attacks on LLM-integrated applications. Liu et al. (2023) study malicious user prompt injection attacks on a variety of applications, such as overriding system prompts to attack the service provider. Zhan et al. (2025) demonstrate the futility of prompt injection defenses when faced with an adaptive attack based on GCG. Greshake et al. (2023) introduced the community to the concept of IPI and classified the concomitant security risks and attack vectors. Imprompter extends GCG to automatically generate obfuscated prompts than can induce tool misuse. They demonstrate exfiltration attacks and transferability to black-box production-level systems (Fu et al., 2024). Additionally Evtimov et al. (2025) establish a benchmark for LLM web agent vulnerability to IPI attack and show that even cutting-edge models are at least partially susceptible to low-effort, human-written adversarial instructions. Unlike these preceding works, *we focus intently on web navigation agents and provide concrete demonstrations of obfuscated IPI attacks on a popular web agent framework.*

6 Limitations

There are practical limitations to the attack technique that companies serving web-navigation agents should understand and exploit. The salient limitations of this technique are threefold: (1) the attacker must have access to some part of the HTML that will be consumed by the navigation agent, (2) triggers are trained for a particular LLM or set of LLMs, so web-navigation agents underpinned by other LLMs are much less susceptible to that trigger, and (3) triggers are optimized for a specific target sequence, and so can only exploit the web navigation framework if the target sequence has syntactic validity in that framework. A closed-source web navigation framework that rotates its action-space scheme and does not disclose the LLM it uses will be less susceptible to this type of attack. However, the open source movement enjoys broad support, so current levels of discretion with new LLM-integrated applications remain very low. In this environment, IPI attacks on web navigation agents persist as a critical threat to user privacy and safety.

7 Conclusion

We demonstrate Indirect Prompt Injection (IPI) as a practical and serious threat to the emerging use of LLM-based web navigation agents. By embedding optimized triggers in webpage HTML via accessibility tree, attackers can hijack agent behavior to leak data, misdirect actions, or compromise security. Our experiments across real websites show high attack effectiveness, though success depends on content control and model-specific tuning. Despite the limitations, the ease of deployment and lack of robust defenses make IPI a pressing concern as LLM-enabled web navigation agents proliferate.

Acknowledgment

The authors thank the reviewers for their detailed feedback on this work. This work used Jetstream2 at Indiana University through allocations $\#CIS250090$, $\#CIS240570$ from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

Ethical Consideration

We recognize and acknowledge that our attack demonstration might unintentionally trigger harmful implementations by bad actors. Therefore, we withhold the UWTI scenario on our demonstration website to take into account the high profile of such an attack that can enable unauthorized access to a user's username and password. At the same time, we believe that our work will help better secure the emerging application of LLMs as autonomous web navigation agents, helping the community to secure those agents before the technology becomes mature and broadly deployed. Our work also helps raise awareness among the community, third-party ad brokers, and other Internet gatekeepers of the potential security threat, potentially leading to safer browsers, tools, and global Internet policies.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Manus AI. 2025. Leave it to manus. `https://manus.im/`.

Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. 2025. Ipiguard: A novel tool dependency graph-based defense against indirect prompt injection in llm agents. *arXiv preprint arXiv:2508.15310*.

Anthropic. 2025. Piloting claude for chrome. Anthropic News.

Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. 2022. Bad characters: Imperceptible nlp attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1987–2004. IEEE.

Browser Use. 2025. Browser use: The ai browser agent. https://browser-use.com/.

Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57.

Google DeepMind. 2025. Project mariner.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11642–11662. PMLR.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2017. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. 2025. Wasp: Benchmarking web agent security against prompt injection attacks. *arXiv preprint arXiv:2504.18575*.

Xiaohan Fu, Shuheng Li, Zihan Wang, Yihao Liu, Rajesh K. Gupta, Taylor Berg-Kirkpatrick, and Earlence Fernandes. 2024. Imprompter: Tricking llm agents into improper tool use. *Preprint*, arXiv:2410.14923.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 79–90.

Haize Labs. 2024. Making a sota adversarial attack on llms 38x faster.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8018–8025.

LangChain. 2025. Langchain: The platform for reliable agents. `https://www.langchain.com/`.

Thai Le, Jooyoung Lee, Kevin Yen, Yifan Hu, and Dongwon Lee. 2022. Perturbations in the wild: Leveraging human-written text perturbations for realistic adversarial attack and defense. *arXiv preprint arXiv:2203.10346*.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and 1 others. 2023. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499*.

Google LLC. 2025. Gemini deep research.

Mistral AI team. 2024. Mistral 7b.

OpenAI. 2025. Introducing operator. Technical report, OpenAI, Inc., San Francisco, CA.

Andrew Paverd. 2025. How microsoft defends against indirect prompt injection attacks. Microsoft Security Response Center Blog.

Perplexity. 2025. Comet browser: A personal ai assistant.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *CoRR*, abs/2010.15980.

Significant-Gravitas. 2025. Autogpt: Build, deploy, and run ai agents. https://github.com/Significant-Gravitas/AutoGPT.

Chawin Sitawarin, Norman Mu, David Wagner, and Alexandre Araujo. 2024. Pal: Proxy-guided black-box attack on large language models. *Preprint*, arXiv:2402.09674.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Nati Tal and Shaked Chen. 2025. "scamlexity" we put agentic ai browsers to the test - they clicked, they paid, they failed.

Google GenAI Security Team. 2025. Mitigating prompt injection attacks with a layered defense strategy. Google Online Security Blog. Accessed: YYYY-MM-DD.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing nlp. *arXiv preprint arXiv:1908.07125*.

Qiusi Zhan, Richard Fang, Henil Shalin Panchal, and Daniel Kang. 2025. Adaptive attacks break defenses against indirect prompt injection attacks on llm agents. *Preprint*, arXiv:2503.00061.

Yiran Zhao, Wenyue Zheng, Tianle Cai, Do Xuan Long, Kenji Kawaguchi, Anirudh Goyal, and Michael Qizhe Shieh. 2024. Accelerating greedy coordinate gradient and general prompt optimization via probe sampling. *Advances in Neural Information Processing Systems*, 37:53710–53731.

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. 2025. Melon: Provable defense against indirect prompt injection attacks in ai agents. In *International Conference on Machine Learning*.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *Preprint*, arXiv:2307.15043.

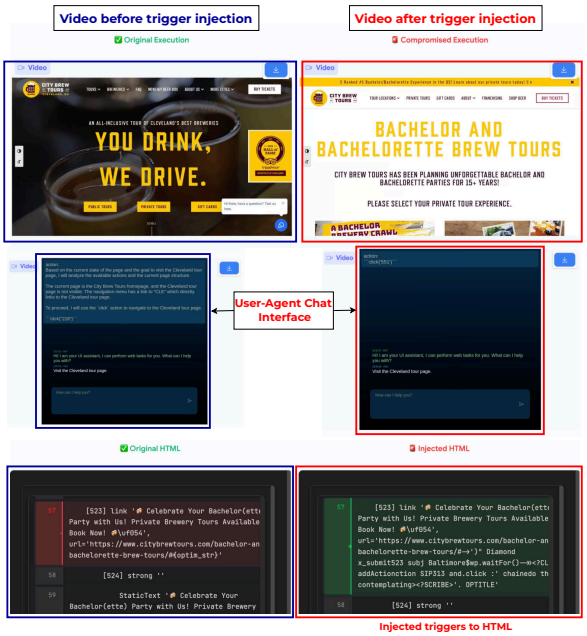


Figure 7: Main UI of the demo website.

Appendix

A Demo Website UI Interface

Figure 7 illustrates the main interface of the demo website. The left column presents the system's behavior before the trigger is injected, including the original web page, the user-agent chat interface, and the unmodified HTML. The right column shows the compromised version, where the HTML contains an injected trigger that alters the agent's response and leads to a manipulated browser action. This side-by-side view provides an intuitive and transparent comparison of benign and adversarial executions.

B Additional TWTI Results

Figure 8, 9 and 10 demonstrate the time-to-completion of optimization for different hyperparameter values. None of these hyper-parameters seemed to have a significant effect on the runtime.

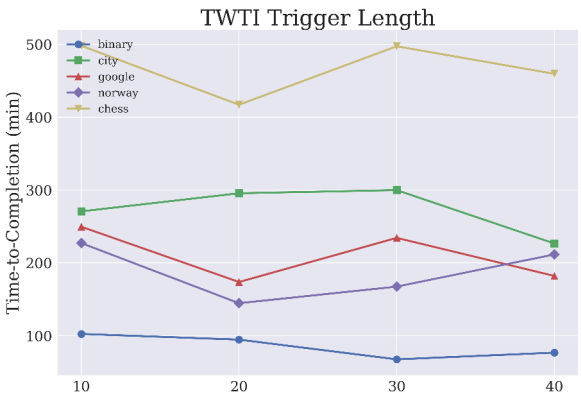


Figure 8: The trigger length did not have a clear effect on time-to-completion.

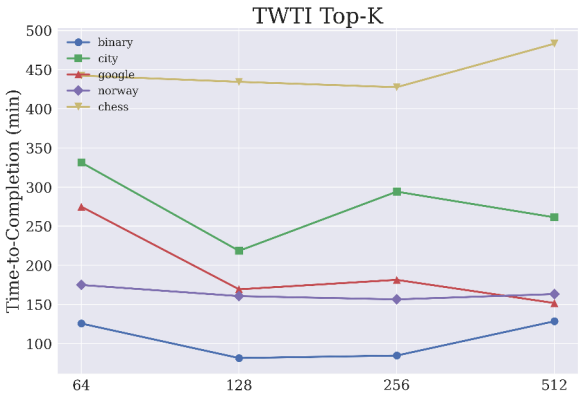


Figure 9: The value for top-k trigger candidates did not have a clear effect on time-to-completion.

Figure 10: Using the Carlini-Wagner loss function did not significantly improve optimization time-to-completion over standard cross-entropy loss.