OWASP Top 10 for LLM

2023

OWASP.ORG/WWW-PROJECT-TOP-10-FOR-LARGE-LANGUAGE-MODEL-APPLICATIONS

Introduction

Welcome to the first iteration of the OWASP Top 10 for Large Language Models (LLMs) Applications.
This document marks an exciting new chapter in the ongoing efforts to enhance security in the rapidly evolving field of artificial intelligence.

A GROUNDBREAKING EFFORT

This initiative is the culmination of the tireless efforts of our first expert team, a diverse group of security specialists, AI researchers, developers, and industry leaders. Since our inception just a month ago, our ranks have swelled to over 370 members, with more than 100 experts actively contributing. This remarkable growth is a testament to the gravity and immediacy of the challenge we face, and the dedication of those working to meet it head on.

The heartiest of congratulations and deepest gratitude is due to our team. The knowledge, time, and passion they have dedicated to this project have been invaluable. This endeavor wouldn't have been possible without their profound insights and unwavering commitment.

OUR PURPOSE

The purpose of our group, as outlined in the OWASP Top 10 for LLM Applications Working Group Charter, is to identify and highlight the top security and safety issues that developers and security teams must consider when building applications leveraging Large Language Models (LLMs). Our objective is to provide clear, practical, and actionable guidance to enable these teams to proactively address potential vulnerabilities in LLM-based applications.

The ultimate aim is to provide a robust foundation for the safe and secure utilization of LLMs across a wide array of scenarios, from small-scale individual projects to large-scale corporate and governmental implementations. We firmly believe that by understanding and mitigating the top vulnerabilities associated with LLMs, we can contribute to a safer and more reliable digital environment for everyone.

ABOUT THIS VERSION

This document, Version 0.5, serves as a crucial milestone in our ongoing journey. It encapsulates the collective insights and understanding of our group, at this early stage, of the unique vulnerabilities inherent to applications leveraging LLMs. It's important to note that this is not the final version of the OWASP Top 10 for LLMs. Instead, consider it a 'preview' of what's to come.

We are committed to refining, expanding, and deepening our work over the coming month as we work towards the true Version 1.0. We invite you to explore this preliminary list, to share your feedback, and to join us in our mission to create a safer, more secure future for AI.

Once again, we extend our heartfelt thanks to our expert team and the broader community for their continued support. Together, let's navigate the exciting and complex world of LLMs with an eye towards security, safety, and inclusivity.

Steve Wilson
Project Lead, OWASP Top 10 for LLM AI Applications
Twitter: @virtualsteve

OWASP Top 10 for LLM

This is a draft list of important vulnerability types for Artificial Intelligence (AI) applications built on Large Language Models (LLMs).

LLM01: Prompt Injections

Prompt Injection Vulnerabilities in LLMs involve crafty inputs leading to undetected manipulations. The impact ranges from data exposure to unauthorized actions, serving attacker's goals.

LLM02: Insecure Output Handling

These occur when plugins or apps accept LLM output without scrutiny, potentially leading to XSS, CSRF, SSRF, privilege escalation, remote code execution, and can enable agent hijacking attacks.

LLM03: Training Data Poisoning

LLMs learn from diverse text but risk training data poisoning, leading to user misinformation. Overreliance on AI is a concern. Key data sources include Common Crawl, WebText, OpenWebText, and books.

LLM04: Denial of Service

An attacker interacts with an LLM in a way that is particularly resource-consuming, causing quality of service to degrade for them and other users, or for high resource costs to be incurred.

LLM05: Supply Chain

LLM supply chains risk integrity due to vulnerabilities leading to biases, security breaches, or system failures. Issues arise from pre-trained models, crowdsourced data, and plugin extensions.

LLM06: Permission Issues

Lack of authorization tracking between plugins can enable indirect prompt injection or malicious plugin usage, leading to privilege escalation, confidentiality loss, and potential remote code execution.

LLM07: Data Leakage

Data leakage in LLMs can expose sensitive information or proprietary details, leading to privacy and security breaches. Proper data sanitization, and clear terms of use are crucial for prevention.

LLM08: Excessive Agency

When LLMs interface with other systems, unrestricted agency may lead to undesirable operations and actions. Like web-apps, LLMs should not self-police; controls must be embedded in APIs.

LLM09: Overreliance

Overreliance on LLMs can lead to misinformation or inappropriate content due to "hallucinations." Without proper oversight, this can result in legal issues and reputational damage.

LLM10: Insecure Plugins

Plugins connecting LLMs to external resources can be exploited if they accept free-form text inputs, enabling malicious requests that could lead to undesired behaviors or remote code execution.

LLM01: Prompt Injection

First Published: July 1st, 2023

Prompt Injection Vulnerabilities occur when attackers manipulate a trusted large language model (LLM) using crafted input prompts, via multiple channels. This manipulation often goes undetected due to inherent trust in LLM's output.

Two types exist: Direct, where the attacker influences the LLM's input, and Indirect, where a 'poisoned' data source affects the LLM.

Outcomes can range from exposing sensitive information to influencing decisions. In complex cases, the LLM could be tricked into unauthorized actions or impersonations, effectively serving the attacker's goals without alerting the user or triggering safeguards.

COMMON VULNERABILITIES

Example 1: An attacker introduces a malicious prompt in a webpage, which is then accessed by the LLM, leading to manipulated responses.

Example 2: The LLM is tricked into divulging sensitive information or manipulating output to downstream systems.

Example 3: An attacker exploits the LLM's interaction with plugins, triggering unauthorized actions like unauthorized purchases, undesired social media posts, deleted emails, etc.

HOW TO PREVENT

Privilege Control: Limit the privileges of an LLM to the least necessary for its functionality. Prevent the LLM from altering the user's state without explicit approval.

Enhanced Input Validation: Implement robust input validation and sanitization methods to filter out potential malicious prompt inputs from untrusted sources.

Segregation and Control of External Content Interaction: Segregate untrusted content from user prompts and control the interaction with external content, especially with plugins that could result in irreversible actions or exposure of personally identifiable information (PII).

Manage Trust: Establish trust boundaries between the LLM, external sources, and extensible functionality (e.g., plugins or downstream functions). Treat the LLM as an untrusted user and maintain final user control on decision making processes.

EXAMPLE ATTACK SCENARIOS

Scenario A: A user employs an LLM to summarize a webpage containing a hidden prompt injection. This then causes the LLM to solicit sensitive information from the user and perform exfiltration via JavaScript or Markdown.

Scenario B: A malicious user uploads a resume with a prompt injection. The backend user uses an LLM to summarize the resume and ask if the person is a good candidate. Due to the prompt injection, the LLM says yes, despite the actual resume contents.

Scenario C: A user enables a plugin linked to an e-commerce site. A rogue instruction embedded on a visited website exploits this plugin, leading to unauthorized purchases.

REFERENCE LINKS

- ChatGPT Plugin Vulnerabilities - Chat with Code
- ChatGPT Cross Plugin Request Forgery and Prompt Injection
- Defending ChatGPT against Jailbreak Attack via Self-Reminder
- Prompt Injection attack against LLM-integrated Applications
- Inject My PDF: Prompt Injection for your Resume
- ChatML for OpenAI API Calls
- Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection
- Threat Modeling LLM Applications
- AI Injections: Direct and Indirect Prompt Injections and Their Implications

LLM02: Insecure Output Handling

First Published: July 1st, 2023

An Insecure Output Handling vulnerability is a type of prompt injection vulnerability that arises when a plugin or application blindly accepts large language model (LLM) output without proper scrutiny and directly passes it to backend, privileged, or client-side functions. Since LLM-generated content can be controlled by prompt input, this behavior is akin to providing users indirect access to additional functionality.

Successful exploitation of an Insecure Output Handling vulnerability can result in XSS and CSRF in web browsers as well as SSRF, privilege escalation, or remote code execution on backend systems. The impact of this vulnerability increases when the application allows LLM content to perform actions above the intended user's privileges. Additionally, this can be combined with agent hijacking attacks to allow an attacker privileged access into a target user's environment.

COMMON VULNERABILITIES

Example 1: LLM output is entered directly into a backend function, resulting in remote code execution. Example 2: JavaScript or Markdown is generated by the LLM and returned to a user. The code is then interpreted by the browser, resulting in XSS

HOW TO PREVENT

Treat the model as any other user and apply proper input validation on responses coming from the model to backend functions. Prevention Step 2: Likewise, encode output coming from the model back to users to mitigate undesired JavaScript or Markdown code interpretations.

REFERENCE LINKS

- Arbitrary Code Execution- Snyk Vulnerability
- ChatGPT Plugin Exploit Explained: From Prompt Injection to Accessing Private Data
- New prompt injection attack on ChatGPT web version. Markdown images can steal your chat data.
- Don't blindly trust LLM responses. Threats to chatbots.
- Threat Modeling LLM Applications

EXAMPLE ATTACK SCENARIOS

Scenario A: An application utilizes an LLM plugin to generate responses for a chatbot feature. However, the application directly passes the LLM-generated response into an internal function responsible for executing system commands without proper validation. This allows an attacker to manipulate the LLM output to execute arbitrary commands on the underlying system, leading to unauthorized access or unintended system modifications.

Scenario B: A user utilizes a website summarizer tool powered by a LLM to generate a concise summary of an article. The website includes a prompt injection instructing the LLM to capture sensitive content from either the website or from the users conversation. From there the LLM can encode the sensitive data and send it out to an attacker-controlled server

Scenario C: A malicious user instructs the LLM to return a JavaScript payload back to a user, without sanitization controls. This can occur either through a sharing a prompt, prompt injected website, or chatbot that accepts prompts from a GET request. The LLM would then return the unsanitized XSS payload back to the user. Without additional filters, outside of those expected by the LLM itself, the JavaScript would execute within the users browser.

LLM03: Training Data Poisioning

First Published: July 1st, 2023

Large language models (LLMs) use diverse raw text to learn and generate outputs. However, training data poisoning, where an attacker introduces vulnerabilities, can compromise the model, exposing users to incorrect information. The OWASP List for LLMs highlights the risk of overreliance on AI content. Key data sources include Common Crawl, used for models like T5 and GPT-3; WebText and OpenWebText, containing public news and Wikipedia; and books, which make up 16% of GPT-3's training data.

COMMON VULNERABILITIES

Example 1: A malicious actor, or a competitor brand intentionally creates inaccurate or malicious documents which are targeted at a model's training data.

- The victim model trains using falsified information which is reflected in outputs of generative AI prompts to it's consumers.

Example 2: In reverse, unintentionally, a model is trained using data which has not been verified by its source, origin or content.

Example 3: The model itself has unrestricted access or inadequate sandboxing to gather datasets to be used as training data which has negative influence on outputs of generative AI prompts as well as loss of control from a management perspective.

HOW TO PREVENT

1. Verify supply chain of the Training Data if sourced externally as well as maintaining attestations, similar to SBOM (Software Bill of Materials) methodology.
2. Prevention Step 2: Verify legitimacy of data sources and data within.
3. Prevention Step 3: Craft different models via separate Training Data for different use-cases to create a more granular and accurate generative AI output.
4. Prevention Step 4: Ensure sufficient sandboxing (LLM03:2023 - Inadequate Sandboxing) is present to prevent the model from scraping unintended data sources.
5. Prevention Step 5: Use strict vetting or input filters for specific Training Data, or categories of data sources to control volume of falsified data.
6. Prevention Step 6: Implement dedicated LLM's to benchmark against undesired consequences and train other LLM's using reinforcement learning techniques.
7. Optional Prevention Step 7: Perform LLM-based red team exercises or LLM vulnerability scanning into the testing phases of the LLM's lifecycle.

EXAMPLE ATTACK SCENARIOS

Scenario A: The LLM generative AI prompt output can mislead users of the application which can lead to unbiased opinions, followings or even worse, hate crimes etc.

Scenario B: If the training data is not correctly filtered, a malicious user of the application may try to influence and inject toxic data into the model for it to adapt to the unbiased and false data.

Scenario C: A malicious actor, or a competitor brand intentionally creates inaccurate or malicious documents which are targeted at a model's training data. The victim model trains using falsified information which is reflected in outputs of generative AI prompts to it's consumers.

REFERENCE LINKS

1. Stanford Research Paper
2. AI Hypocrisy: OpenAI, Google, and Anthropic train using others' content, but wont let others use their content
3. Inject My PDF: Prompt Injection for your Resume
4. How data poisoning attacks corrupt machine learning models
5. LLM10:2023 - Training Data Poisoning

LLM04: Denial of Service

First Published: July 1st, 2023

An attacker interacts with an LLM in a way that is particularly resource-consuming, causing quality of service to degrade for them and other users, or for high resource costs to be incurred.

COMMON VULNERABILITIES

Example 1: Posing queries that lead to recurring resource usage through high-volume generation of tasks in a queue, e.g. with LangChain or AutoGPT

Example 2: Sending queries that are unusually resource-consuming, perhaps because they use unusual orthography or sequences

HOW TO PREVENT

1. Cap resource use per request
2. Cap resource use per step, so that requests involving complex parts execute more slowly
3. Limit the number of queued actions and the number of total actions in a system reacting to LLM responses

EXAMPLE ATTACK SCENARIOS

Scenario A: An attacker repeatedly sends multiple requests to a hosted model that are difficult and costly for it to process. Many resources are allocated, leading to worse service for other users and increased resource bills for the host.

Scenario B: A piece of text on a webpage is encountered while an LLM-driven tool is collecting information to respond to a benign query. That piece of text leads to the tool making many more web page requests. The query ends up leading to large amounts of resource consumption, and receives a slow or even absent response, as do any other queries from similar systems that end up encountering the given piece of text.

REFERENCE LINKS

1. LangChain max_iterations
2. Sponge Examples: Energy-Latency Attacks on Neural Networks

LLM05: Supply Chain

First Published: July 1st, 2023

The supply chain in LLMs can be vulnerable impacting the integrity of training data, ML models, deployment platfoms and lead to biased outcomes, security breaches, or complete system failures. Traditionally, vulnerabilities focused on software components but is extended in AI because of the popularity of transfer learning, re-use of pre-trained models, as well as crowdsourced data. In public LLMs such as OpenGPT extension plugins are also an area susceptible to this vulnerability.

COMMON VULNERABILITIES

Example 1: Use of Vulnerable Model used for Transfer Learning

Example 2: Use of poisoned crowd-sourced data

Example 3: Tampered model or data by an out-sourcing supplier

Example 4: Traditional third-paty Package Vulnerabilities

HOW TO PREVENT

1. Careful Vetting of sources and suppliers
2. Vulnerabilities scanning of components, not only when deploying to production but before used in development and testing; model development environments
3. Use own curated package repositories with vulnerability checks
4. Code Signing
5. Avdersarial robustness tests on supplied models and data for tampering and poisoning and throughout the MLOps pipeline
6. Implement adversarial robustness training to help detect extraction queries.
7. Review and Monitor Supplier Security and Access
8. Auditing

REFERENCE LINKS

- ChatGPT Data Breach Confirmed as Security Firm Warns of Vulnerable Component Exploitation
- What Happens When an AI Company Falls Victim to a Software Supply Chain Vulnerability
- Plugin review process
- Compromised PyTorch-nightly dependency chain
- Failure Modes in Machine Learning
- ML Supply Chain Compromise
- Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples
- BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain
- VirusTotal Poisoning

EXAMPLE ATTACK SCENARIOS

Scenario A: An attacker exploits a vulnerable package to compromise a system. The recent OpenAI breach, was due to a vulnerable third party package

Scenario B: An attacker exploits a malicious or vulnerable ChatGPT plugin to exfiltrate data, bypass restriction,s execute code, span a user or produce malicious content. Although this is a supply-chain vulnerability, because of the ChatGPT plugun integration models it is also covered in Insecure Plugin Integration

Scenario C: An attacker exploits the PyPi package registry to trick model developers to download a compromised package and exfiltrate data or escalate privilege in a model development environment.

Scenario D: An attacker poisons or tampers a copy of publicly available model pre-built model (e.g. LlaMa) to create a backdoor or trojan horse deploys it for others to use either directly or for transfer learning

Scenario E: An attacker poisons or tampers a copy of publicly available data set (e.g. Kaggle) to help create a backdoor or trojan horse in other models.

LLM06: Permission Issues

First Published: July 1st, 2023

Authorization is not tracked between plugins, allowing a malicious actor to take action in the context of the LLM user via indirect prompt injection, use of malicious plugins, or other methods. This can result in privilege escalation, loss of confidentiality, and even remote code execution, depending on available plugins.

COMMON VULNERABILITIES

Example 1: Authentication is performed without explicit authorization to a particular plugin.

Example 2: A plugin treats all LLM content as being created entirely by the user, and performs any requested actions without requiring additional authorization.

Example 3: Plugins are chained together without considering the authorization of one plugin to perform an action using another plugin.

HOW TO PREVENT

1. Require manual authorization of any action taken by sensitive plugins.
2. Call no more than one plugin with each user input, resetting any plugin-supplied data between calls.
3. Prevent sensitive plugins from being called after any other plugin.
4. Perform taint tracing on all plugin content, ensuring that plugin is called with an authorization level corresponding to the lowest authorization of any plugin that has provided input to the LLM prompt.

EXAMPLE ATTACK SCENARIOS

Scenario A: Indirect prompt injection is used to induce an email plugin to deliver the contents of the current user's inbox to a malicious URL via POST request.

Scenario B: An attacker uses indirect prompt injection to abuse a Slack integration, sending a Slack message to @everyone in all available Slacks with an obscene and defamatory comment.

REFERENCE LINKS

1. Plugin Vulnerabilities: Visit a Website and Have Your Source Code Stolen
2. ChatGPT Plugin Exploit Explained: From Prompt Injection to Accessing Private Data

LLM07: Data Leakage

First Published: July 1st, 2023

Data leakage occurs when an LLM accidentally reveals sensitive information, proprietary algorithms, or other confidential details through its responses. This can result in unauthorized access to sensitive data or intellectual property, privacy violations, and other security breaches. It's important to note that users of an LLM application should be aware of how to interact with an LLM and identify the risks present on how they might unintentionally input sensitive data. Vice versa, an LLM application should perform adequate data sanitization and scrubbing validation in aid to prevent user data from entering the training model data. Furthermore, the company hosting the LLM should have appropriate Terms of User policies available to make users aware on how data is processed.

COMMON VULNERABILITIES

Example 1: Incomplete or improper filtering of sensitive information in the LLM's responses.

Example 2: Overfitting or memorization of sensitive data in the LLM's training process.

Example 3: Unintended disclosure of confidential information due to LLM misinterpretation, lack of data scrubbing methods or errors.

HOW TO PREVENT

1. Integrate adequate data sanitization and scrubbing techniques in aid to prevent user data from entering the training model data.
2. Implement robust input validation and sanitization methods to identify and filter out potential malicious inputs.
3. Maintain ongoing supply chain mitigation of risk through techniques such as SAST (Static Application Security Testing) and SBOM (Software Bill of Materials) attestations to identify and remediate vulnerabilities in dependencies for third-party software or packages.
4. Implement dedicated LLM's to benchmark against undesired consequences and train other LLM's using reinforcement learning techniques.
5. Perform LLM-based red team exercises or LLM vulnerability scanning into the testing phases of the LLM's lifecycle.

EXAMPLE ATTACK SCENARIOS

Scenario A: Unsuspecting legitimate user A is exposed to certain other user data via the LLM when interacting with the LLM application in a non-malicious manner.

Scenario B: User A targets a well crafted set of prompts to bypass input filters and sanitization from the LLM to cause it to reveal sensitive information (I.E PII) about other users of the application.

Scenario C: Personal data such as PII is leaked into the model via training data due to either negligence from the user themselves, or the LLM application. This case could increase risk and probability of scenario 1 or 2 above.

REFERENCE LINKS

1. AI data leak crisis: New tool prevents company secrets from being fed to ChatGPT
2. Lessons learned from ChatGPT's Samsung leak
3. Cohere - Terms Of Use

LLM08: Excessive Agency

First Published: July 1st, 2023

An LLM may be granted a degree of agency - the ability to interface with other systems in order to undertake actions. Without restriction, any undesireable operation of the LLM (regardless of the root casue; e.g., halucination, direct/indirect prompt injection, or just poorly-engineered benign prompts, etc) may result in undesireable actions being taken. Just like we never trust client-side validation in web-apps, LLMs should not be trusted to self-police or self-restrict; controls should be embedded in the APIs of the systems being interefaced with.

COMMON VULNERABILITIES

Example 1: Undesireable Actions Performed: The LLM triggers actions outside of the LLM that are unintended or undesireable, leading to second order consequences on downstream systems and processes.

HOW TO PREVENT

1. Reduce the permissions granted to the LMM to the minimum necessary to limit the scope of undesirable actions.
2. Implement rate-limiting to reduce the number of undesirable actions.
3. Utilize human-in-the-loop control to require a human to approve all actions before they are taken.

EXAMPLE ATTACK SCENARIOS

Scenario A: A personal assistant LLM is granted access to an individuals's mailbox in order to summarise the content of incoming emails. The LMM is vulnerable to an indirect promot injection attack, whereby a maliciously-crafted incoming email tricks the LLM into sending spam messages from the user's mailbox. This could be avoided by only granting the LLM read-only access to the mailbox (not the ability to send messages), or by requiring the user to manually review and hit 'send' on every mail drafted by the LLM. Alternatively The damage caused could be reduced by implementing rate limiting on the mail-sending interface.

Scenario B: A customer service LLM has an interface to a payments system to provide service credits or refunds to customers in the case of complaints. The system prompt instructs the LLM to limit refunds to no more than one month's subscription fee, however a malicious customer engineers a direct prompt injection attack to convince the LMM to issue a refund of 100 years of fees. This could be avoided by implementing the 'one month max' limit within the refund API, rather than relying on the LMM to honour the limit in it's system prompt.

LLM09: Overreliance

First Published: July 1st, 2023

Overreliance on LLMs is a security vulnerability that arises when systems excessively depend on LLMs for decision-making or content generation without adequate oversight, validation mechanisms, or risk communication. LLMs, while capable of generating creative and informative content, are also susceptible to "hallucinations," producing content that is factually incorrect, nonsensical, or inappropriate. These hallucinations can lead to misinformation, miscommunication, potential legal issues, and damage to a company's reputation if unchecked.

COMMON VULNERABILITIES

Factually Incorrect Information: An LLM provides incorrect information as a response, leading to misinformation. For example, an LLM may inaccurately describe historical events, resulting in misleading outputs.

Nonsensical Outputs: An LLM generates grammatically correct but logically incoherent or nonsensical text. For instance, the LLM might generate a poem or a story that doesn't make logical sense.

Source Conflation: LLM conflates information from different sources, creating misleading content. It might combine historical data with current events in an incorrect manner.

Overindulgence: LLM might generate an output that could incorrectly be seen as disclosure of confidential information.

Inadequate Risk Communication: Tech companies fail to adequately communicate the inherent risks of using LLMs to the end users, leading to potential negative consequences.

HOW TO PREVENT

1. **Continuous Monitoring:** Regularly monitor and review the outputs of the LLM to ensure they are factual, coherent, and appropriate. Use manual reviews or automated tools for larger scale applications.
2. **Fact Checking:** Verify the accuracy of information provided by LLMs before using it for decision-making, information dissemination, or other critical functions.
3. **Model Tuning:** Tune your LLM to reduce the likelihood of hallucinations. Techniques can include prompt engineering, parameter efficient tuning (PET), and full model tuning.
4. **Set Up Validation Mechanisms:** Implement automatic validation mechanisms to check the generated output against known facts or data.
5. **Improve Risk Communication:** Follow best practices from risk communication literature and lessons from other sectors to facilitate dialogue with users, establish actionable risk communication, and measure the effectiveness of risk communications on an ongoing basis.

EXAMPLE ATTACK SCENARIOS

Scenario A: A corporation uses an LLM to generate customer-facing content. Due to a hallucination, the LLM generates incorrect information about a product, leading to customer confusion, potential loss of sales, and damage to the company's reputation.

Scenario B: An LLM is used in a news organization to assist in generating news articles. The LLM conflates information from different sources and produces an article with misleading information, leading to the dissemination of misinformation and potential legal consequences.

Scenario C: A user communicates with an AI chatbot based on an LLM. The user, unaware of the limitations and risks of the AI, acts on harmful content generated by the model due to the lack of effective risk communication from the tech company.

REFERENCE LINKS

1. How Should Companies Communicate the Risks of Large Language Models to Users?
2. Understanding LLM Hallucinations

LLM10: Insecure Plugins

First Published: July 1st, 2023

A plugin designed to connect an LLM to some external resource accepts free-form text as an input instead of parameterized and type-checked inputs. This allows a potential attacker significant latitude to construct a malicious request to the plugin that could result in a wide range of undesired behaviors, up to and including remote code execution.

COMMON VULNERABILITIES

A plugin designed to call a specific API hosted at a specific endpoint accepts a string containing the entire URL to be retrieved, instead of query parameters to be inserted into the URL.

A plugin designed to look up information from a SQL database accepts a raw SQL query rather than paramters to be inserted into a fully parameterized query.

HOW TO PREVENT

1. Plugin calls should be strictly parameterized wherever possible, including type and range checks on inputs
2. When freeform input must be accepted, it should be carefully inspected to ensure that no potentially harmful methods are being called.
3. The plugin should be designed from a least-privilege perspective, exposing as little functionality as possible while still performing its desired function.

EXAMPLE ATTACK SCENARIOS

Scenario: A plugin prompt provides a base URL and instructs the LLM to combine the URL with a query to obtain weather forecasts in response to user requests. The resulting URL is then accessed and the results shown to the user. A malicious user crafts a request such that a URL pointing to a domain they control, and not the URL hosting the weather service API, is accessed, allowing the malicious user to obtain the IP address of the plugin for further reconnaisance, as well as to inject their own text into the LLM system via their domain, potentially granting them further access to downstream plugins.

Core Team & Contributors

Core Team Members are listed in Blue

Adrian Culley	Trellix
Ads Dawson	Cohere
Alexander Zai	
Amay Trivedi	
Andrea Succi	
Andrew Amaro	Klavan Security Group
Andy Dyrcz	Linkfire
Andy Smith	
Ananda Krishna	Astra Security
Ashish Rajan	Cloud Security Podcast
Autumn Moulder	
Bilal Siddiqui	Trustwave
Brian Pendleton	AVID
Brodie McRae	AWS
Dan Frommer	
David Rowe	
David Taylor	
Dr. Matteo Große-Kampmann	AWARE7
Emanuel Valente	iFood
Emmanuel Guilherme Junior	McMaster University
Eugene Neelou	
Eugene Tawiah	Complex Technologies
Guillaume Ehinger	Google
Gavin Klondike	AI Village
Itamar Golan	
James Rabe	IriusRisk
Jason Haddix	Buddobot
Jason Ross	Salesforce
Jeff Williams	Contrast Security
Jinson Varghese Behanan	Astra Security
John Sotiropoulos	Kainos
Joshua Nussbaum	
Kai Greshake	
Kelvin Low	Aigos
Ken Arora	F5
Ken Huang	DistributedApps.ai
Larry Carson	
Leon Derczynski	U of Washington, IT U of Copenhagen
Leonardo Shikida	IBM
Lior Drihem	
Manjesh S	HackerOne
Mike Finch	HackerOne
Nathan Hamiel	Kudelski Security
Nir Paz	
Otto Sulin	Nordic Venture Family
Patrick Biyaga	Thenavigo
Priyadharshini Parthasarathy	Coalfire
Rahul Zhade	GitHub
Rachit Sood	
Reza Rashidi	HADESS
Rich Harang	
Santosh Kumar	Cisco
Sarah Thornton	Red Hat
Steve Wilson	Contrast Security
Vandana Verma Sehgal	Snyk
Vinay Vishwanatha	Sprinklr
Vladimir Fedotov	EPAM
Vishwas Manral	
Will Chilcutt	Yahoo