# CWE Detail – CWE-1427

## Description

The product uses externally-provided data to build prompts provided to  
large language models (LLMs), but the way these prompts are constructed  
causes the LLM to fail to distinguish between user-supplied inputs and  
developer provided system directives.

## Extended Description

When prompts are constructed using externally controllable data, it is  
often possible to cause an LLM to ignore the original guidance provided by  
its creators (known as the "system prompt") by inserting malicious  
instructions in plain human language or using bypasses such as special  
characters or tags. Because LLMs are designed to treat all instructions as  
legitimate, there is often no way for the model to differentiate between  
what prompt language is malicious when it performs inference and returns  
data. Many LLM systems incorporate data from other adjacent products or  
external data sources like Wikipedia using API calls and retrieval  
augmented generation (RAG). Any external sources in use that may contain  
untrusted data should also be considered potentially malicious.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2023-32786: Chain: LLM integration framework has prompt injection  
 (CWE-1427) that allows an attacker to force the service to retrieve  
 data from an arbitrary URL, essentially providing SSRF (CWE-918) and  
 potentially injecting content into downstream tasks.

**•** CVE-2024-5184: ML-based email analysis product uses an  
 API service that allows a malicious user to inject a  
 direct prompt and take over the service logic, forcing  
 it to leak the standard hard-coded system prompts  
 and/or execute unwanted prompts to leak sensitive  
 data.

**•** CVE-2024-5565: Chain: library for generating SQL via LLMs using RAG uses  
 a prompt function to present the user with visualized results,  
 allowing altering of the prompt using prompt injection (CWE-1427) to  
 run arbitrary Python code (CWE-94) instead of the intended  
 visualization code.

## Modes of Introduction

**•** Architecture and Design: LLM-connected applications that do not distinguish between  
trusted and untrusted input may introduce this weakness. If such  
systems are designed in a way where trusted and untrusted instructions  
are provided to the model for inference without differentiation, they  
may be susceptible to prompt injection and similar attacks.

**•** Implementation: When designing the application, input validation should be  
applied to user input used to construct LLM system prompts. Input  
validation should focus on mitigating well-known software security  
risks (in the event the LLM is given agency to use tools or perform  
API calls) as well as preventing LLM-specific syntax from being  
included (such as markup tags or similar).

**•** Implementation: This weakness could be introduced if training does not account  
for potentially malicious inputs.

**•** System Configuration: Configuration could enable model parameters to be manipulated  
when this was not intended.

**•** Integration: This weakness can occur when integrating the model into the software.

**•** Bundling: This weakness can occur when bundling the model with the software.

## Common Consequences

**•** Impact: Execute Unauthorized Code or Commands, Varies by Context — Notes:

**•** Impact: Read Application Data — Notes:

**•** Impact: Modify Application Data, Execute Unauthorized Code or Commands — Notes:

**•** Impact: Read Application Data, Modify Application Data, Gain Privileges or Assume Identity — Notes:

## Potential Mitigations

**•** Architecture and Design: LLM-enabled applications should be designed to ensure  
proper sanitization of user-controllable input, ensuring that no  
intentionally misleading or dangerous characters can be  
included. Additionally, they should be designed in a way that ensures  
that user-controllable input is identified as untrusted and  
potentially dangerous. (Effectiveness: High)

**•** Implementation: LLM prompts should be constructed in a way that  
effectively differentiates between user-supplied input and  
developer-constructed system prompting to reduce the chance of model  
confusion at inference-time. (Effectiveness: Moderate)

**•** Architecture and Design: LLM-enabled applications should be designed to ensure  
proper sanitization of user-controllable input, ensuring that no  
intentionally misleading or dangerous characters can be  
included. Additionally, they should be designed in a way that ensures  
that user-controllable input is identified as untrusted and  
potentially dangerous. (Effectiveness: High)

**•** Implementation: Ensure that model training includes training examples  
that avoid leaking secrets and disregard malicious inputs. Train the  
model to recognize secrets, and label training data  
appropriately. Note that due to the non-deterministic nature of  
prompting LLMs, it is necessary to perform testing of the same test  
case several times in order to ensure that troublesome behavior is not  
possible. Additionally, testing should be performed each time a new  
model is used or a model's weights are updated. (Effectiveness: N/A)

**•** Installation: During deployment/operation, use components that operate externally to the system to  
monitor the output and act as a moderator. These components are called  
different terms, such as supervisors or guardrails. (Effectiveness: N/A)

**•** System Configuration: During system configuration, the model could be  
fine-tuned to better control and neutralize potentially dangerous  
inputs. (Effectiveness: N/A)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** To avoid XSS risks, the code ensures that the response from the chatbot is properly encoded for HTML output. If the user provides CWE-77 and CWE-78, then the resulting prompt would look like:

**•** This agent is provided minimal context on how to treat dangerous  
 requests for a secret. Suppose the user provides an input like: