# CWE Detail – CWE-74

## Description

The product constructs all or part of a command, data structure, or record using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify how it is parsed or interpreted when it is sent to a downstream component.

## Extended Description

Software or other automated logic has certain assumptions about what constitutes data and control respectively. It is the lack of verification of these assumptions for user-controlled input that leads to injection problems. Injection problems encompass a wide variety of issues -- all mitigated in very different ways and usually attempted in order to alter the control flow of the process. For this reason, the most effective way to discuss these weaknesses is to note the distinct features that classify them as injection weaknesses. The most important issue to note is that all injection problems share one thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2024-5184: API service using a large generative AI model allows direct prompt injection to leak hard-coded system prompts or execute other prompts.

**•** CVE-2022-36069: Python-based dependency management tool avoids OS command injection when generating Git commands but allows injection of optional arguments with input beginning with a dash (CWE-88), potentially allowing for code execution.

**•** CVE-1999-0067: Canonical example of OS command injection. CGI program does not neutralize "|" metacharacter when invoking a phonebook program.

**•** CVE-2022-1509: injection of sed script syntax ("sed injection")

**•** CVE-2020-9054: Chain: improper input validation (CWE-20) in username parameter, leading to OS command injection (CWE-78), as exploited in the wild per CISA KEV. (KEV)

**•** CVE-2021-44228: Product does not neutralize ${xyz} style expressions, allowing remote code execution. (log4shell vulnerability) (KEV)

## Related Attack Patterns (CAPEC)

* CAPEC-10
* CAPEC-101
* CAPEC-105
* CAPEC-108
* CAPEC-120
* CAPEC-13
* CAPEC-135
* CAPEC-14
* CAPEC-24
* CAPEC-250
* CAPEC-267
* CAPEC-273
* CAPEC-28
* CAPEC-3
* CAPEC-34
* CAPEC-42
* CAPEC-43
* CAPEC-45
* CAPEC-46
* CAPEC-47
* CAPEC-51
* CAPEC-52
* CAPEC-53
* CAPEC-6
* CAPEC-64
* CAPEC-67
* CAPEC-7
* CAPEC-71
* CAPEC-72
* CAPEC-76
* CAPEC-78
* CAPEC-79
* CAPEC-8
* CAPEC-80
* CAPEC-83
* CAPEC-84
* CAPEC-9

## Attack TTPs

**•** T1574.007: Path Interception by PATH Environment Variable (Tactics: persistence, privilege-escalation, defense-evasion)

**•** T1574.006: Dynamic Linker Hijacking (Tactics: persistence, privilege-escalation, defense-evasion)

**•** T1562.003: Impair Command History Logging (Tactics: defense-evasion)

**•** T1027: Obfuscated Files or Information (Tactics: defense-evasion)

## Modes of Introduction

**•** Implementation: REALIZATION: This weakness is caused during implementation of an architectural security tactic.

## Common Consequences

**•** Impact: Read Application Data — Notes: Many injection attacks involve the disclosure of important information -- in terms of both data sensitivity and usefulness in further exploitation.

**•** Impact: Bypass Protection Mechanism — Notes: In some cases, injectable code controls authentication; this may lead to a remote vulnerability.

**•** Impact: Alter Execution Logic — Notes: Injection attacks are characterized by the ability to significantly change the flow of a given process, and in some cases, to the execution of arbitrary code.

**•** Impact: Other — Notes: Data injection attacks lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing.

**•** Impact: Hide Activities — Notes: Often the actions performed by injected control code are unlogged.

## Potential Mitigations

**•** Requirements: Programming languages and supporting technologies might be chosen which are not subject to these issues. (Effectiveness: N/A)

**•** Implementation: Utilize an appropriate mix of allowlist and denylist parsing to filter control-plane syntax from all input. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** The $userName variable is not checked for malicious input. An attacker could set the $userName variable to an arbitrary OS command such as:

**•** Assuming a string consisting of standard alpha-numeric characters, such as "Jane Smith", is submitted in the request the HTTP response including this cookie might take the following form:

**•** However, validate\_name() alows  
 filenames that begin with a "-". An adversary could  
 supply a filename like "-aR", producing the "ls -l -aR"  
 command (CWE-88), thereby getting a full recursive  
 listing of the entire directory and all of its  
 sub-directories. There are a couple possible mitigations for this  
 weakness. One would be to refactor the code to avoid  
 using system() altogether, instead relying on internal  
 functions. Another option could be to add a "--" argument  
 to the ls command, such as "ls -l --", so that any  
 remaining arguments are treated as filenames, causing  
 any leading "-" to be treated as part of a filename  
 instead of another option. Another fix might be to change the regular expression used in validate\_name to force the first character of the filename to be a letter or number, such as:

**•** 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:

## Notes

**•** Theoretical: Many people treat injection only as an input validation problem (CWE-20) because many people do not distinguish between the consequence/attack (injection) and the protection mechanism that prevents the attack from succeeding. However, input validation is only one potential protection mechanism (output encoding is another), and there is a chaining relationship between improper input validation and the improper enforcement of the structure of messages to other components. Other issues not directly related to input validation, such as race conditions, could similarly impact message structure.