# CWE Detail – CWE-138

## Description

The product receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as control elements or syntactic markers when they are sent to a downstream component.

## Extended Description

Most languages and protocols have their own special elements such as characters and reserved words. These special elements can carry control implications. If product does not prevent external control or influence over the inclusion of such special elements, the control flow of the program may be altered from what was intended. For example, both Unix and Windows interpret the symbol < ("less than") as meaning "read input from a file".

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2001-0677: Read arbitrary files from mail client by providing a special MIME header that is internally used to store pathnames for attachments.

**•** CVE-2000-0703: Setuid program does not cleanse special escape sequence before sending data to a mail program, causing the mail program to process those sequences.

**•** CVE-2003-0020: Multi-channel issue. Terminal escape sequences not filtered from log files.

**•** CVE-2003-0083: Multi-channel issue. Terminal escape sequences not filtered from log files.

## Related Attack Patterns (CAPEC)

* CAPEC-105
* CAPEC-15
* CAPEC-34

## Modes of Introduction

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

## Common Consequences

**•** Impact: Execute Unauthorized Code or Commands, Alter Execution Logic, DoS: Crash, Exit, or Restart — Notes:

## Potential Mitigations

**•** Implementation: Developers should anticipate that special elements (e.g. delimiters, symbols) will be injected into input vectors of their product. One defense is to create an allowlist (e.g. a regular expression) that defines valid input according to the requirements specifications. Strictly filter any input that does not match against the allowlist. Properly encode your output, and quote any elements that have special meaning to the component with which you are communicating. (Effectiveness: N/A)

**•** Implementation: Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. (Effectiveness: N/A)

**•** Implementation: Use and specify an appropriate output encoding to ensure that the special elements are well-defined. A normal byte sequence in one encoding could be a special element in another. (Effectiveness: N/A)

**•** Implementation: Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked. (Effectiveness: N/A)

**•** Implementation: While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88). (Effectiveness: N/A)

## Applicable Platforms

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

## Notes

**•** Relationship: This weakness can be related to interpretation conflicts or interaction errors in intermediaries (such as proxies or application firewalls) when the intermediary's model of an endpoint does not account for protocol-specific special elements.

**•** Relationship: See this entry's children for different types of special elements that have been observed at one point or another. However, it can be difficult to find suitable CVE examples. In an attempt to be complete, CWE includes some types that do not have any associated observed example.

**•** Research Gap: This weakness is probably under-studied for proprietary or custom formats. It is likely that these issues are fairly common in applications that use their own custom format for configuration files, logs, meta-data, messaging, etc. They would only be found by accident or with a focused effort based on an understanding of the format.