# CWE Detail – CWE-150

## Description

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

## Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2002-0542: The mail program processes special "~" escape sequence even when not in interactive mode.

**•** CVE-2000-0703: Setuid program does not filter escape sequences before calling mail program.

**•** CVE-2002-0986: Mail function does not filter control characters from arguments, allowing mail message content to be modified.

**•** 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.

**•** CVE-2003-0021: Terminal escape sequences not filtered by terminals when displaying files.

**•** CVE-2003-0022: Terminal escape sequences not filtered by terminals when displaying files.

**•** CVE-2003-0023: Terminal escape sequences not filtered by terminals when displaying files.

**•** CVE-2003-0063: Terminal escape sequences not filtered by terminals when displaying files.

**•** CVE-2000-0476: Terminal escape sequences not filtered by terminals when displaying files.

**•** CVE-2001-1556: MFV. (multi-channel). Injection of control characters into log files that allow information hiding when using raw Unix programs to read the files.

## Related Attack Patterns (CAPEC)

* CAPEC-134
* CAPEC-41
* CAPEC-81
* CAPEC-93

## Modes of Introduction

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

## Common Consequences

**•** Impact: Unexpected State — Notes:

## Potential Mitigations

**•** N/A: Developers should anticipate that escape, meta and control characters/sequences will be injected/removed/manipulated in the input vectors of their product. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system. (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: 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)

**•** 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)

## Applicable Platforms

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