# CWE Detail – CWE-838

## Description

The product uses or specifies an encoding when generating output to a downstream component, but the specified encoding is not the same as the encoding that is expected by the downstream component.

## Extended Description

This weakness can cause the downstream component to use a decoding method that produces different data than what the product intended to send. When the wrong encoding is used - even if closely related - the downstream component could decode the data incorrectly. This can have security consequences when the provided boundaries between control and data are inadvertently broken, because the resulting data could introduce control characters or special elements that were not sent by the product. The resulting data could then be used to bypass protection mechanisms such as input validation, and enable injection attacks. While using output encoding is essential for ensuring that communications between components are accurate, the use of the wrong encoding - even if closely related - could cause the downstream component to misinterpret the output. For example, HTML entity encoding is used for elements in the HTML body of a web page. However, a programmer might use entity encoding when generating output for that is used within an attribute of an HTML tag, which could contain functional Javascript that is not affected by the HTML encoding. While web applications have received the most attention for this problem, this weakness could potentially apply to any type of product that uses a communications stream that could support multiple encodings.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2009-2814: Server does not properly handle requests that do not contain UTF-8 data; browser assumes UTF-8, allowing XSS.

## Related Attack Patterns (CAPEC)

* CAPEC-468

## Common Consequences

**•** Impact: Modify Application Data, Execute Unauthorized Code or Commands — Notes: An attacker could modify the structure of the message or data being sent to the downstream component, possibly injecting commands.

## Potential Mitigations

**•** Implementation: Use context-aware encoding. That is, understand which encoding is being used by the downstream component, and ensure that this encoding is used. If an encoding can be specified, do so, instead of assuming that the default encoding is the same as the default being assumed by the downstream component. (Effectiveness: N/A)

**•** Architecture and Design: Where possible, use communications protocols or data formats that provide strict boundaries between control and data. If this is not feasible, ensure that the protocols or formats allow the communicating components to explicitly state which encoding/decoding method is being used. Some template frameworks provide built-in support. (Effectiveness: N/A)

**•** Architecture and Design: Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using the ESAPI Encoding control [REF-45] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error. Note that some template mechanisms provide built-in support for the appropriate encoding. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** The programmer attempts to avoid XSS exploits (CWE-79) by encoding the POST values so they will not be interpreted as valid HTML. However, the htmlentities() encoding is not appropriate when the data are used as HTML attributes, allowing more attributes to be injected.