# CWE Detail – CWE-116

## Description

The product prepares a structured message for communication with another component, but encoding or escaping of the data is either missing or done incorrectly. As a result, the intended structure of the message is not preserved.

## Extended Description

Improper encoding or escaping can allow attackers to change the commands that are sent to another component, inserting malicious commands instead. Most products follow a certain protocol that uses structured messages for communication between components, such as queries or commands. These structured messages can contain raw data interspersed with metadata or control information. For example, "GET /index.html HTTP/1.1" is a structured message containing a command ("GET") with a single argument ("/index.html") and metadata about which protocol version is being used ("HTTP/1.1"). If an application uses attacker-supplied inputs to construct a structured message without properly encoding or escaping, then the attacker could insert special characters that will cause the data to be interpreted as control information or metadata. Consequently, the component that receives the output will perform the wrong operations, or otherwise interpret the data incorrectly.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2021-41232: Chain: authentication routine in Go-based agile development product does not escape user name (CWE-116), allowing LDAP injection (CWE-90)

**•** CVE-2008-4636: OS command injection in backup software using shell metacharacters in a filename; correct behavior would require that this filename could not be changed.

**•** CVE-2008-0769: Web application does not set the charset when sending a page to a browser, allowing for XSS exploitation when a browser chooses an unexpected encoding.

**•** CVE-2008-0005: Program does not set the charset when sending a page to a browser, allowing for XSS exploitation when a browser chooses an unexpected encoding.

**•** CVE-2008-5573: SQL injection via password parameter; a strong password might contain "&"

**•** CVE-2008-3773: Cross-site scripting in chat application via a message subject, which normally might contain "&" and other XSS-related characters.

**•** CVE-2008-0757: Cross-site scripting in chat application via a message, which normally might be allowed to contain arbitrary content.

## Related Attack Patterns (CAPEC)

* CAPEC-104
* CAPEC-73
* CAPEC-81
* CAPEC-85

## Modes of Introduction

**•** Implementation: N/A

**•** Operation: N/A

## Common Consequences

**•** Impact: Modify Application Data — Notes: The communications between components can be modified in unexpected ways. Unexpected commands can be executed, bypassing other security mechanisms. Incoming data can be misinterpreted.

**•** Impact: Execute Unauthorized Code or Commands — Notes: The communications between components can be modified in unexpected ways. Unexpected commands can be executed, bypassing other security mechanisms. Incoming data can be misinterpreted.

**•** Impact: Bypass Protection Mechanism — Notes: The communications between components can be modified in unexpected ways. Unexpected commands can be executed, bypassing other security mechanisms. Incoming data can be misinterpreted.

## Potential Mitigations

**•** 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. Alternately, use built-in functions, but consider using wrappers in case those functions are discovered to have a vulnerability. (Effectiveness: N/A)

**•** Architecture and Design: If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. For example, stored procedures can enforce database query structure and reduce the likelihood of SQL injection. (Effectiveness: N/A)

**•** Architecture and Design: Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. (Effectiveness: N/A)

**•** Architecture and Design: In some cases, input validation may be an important strategy when output encoding is not a complete solution. For example, you may be providing the same output that will be processed by multiple consumers that use different encodings or representations. In other cases, you may be required to allow user-supplied input to contain control information, such as limited HTML tags that support formatting in a wiki or bulletin board. When this type of requirement must be met, use an extremely strict allowlist to limit which control sequences can be used. Verify that the resulting syntactic structure is what you expect. Use your normal encoding methods for the remainder of the input. (Effectiveness: N/A)

**•** Architecture and Design: Use input validation as a defense-in-depth measure to reduce the likelihood of output encoding errors (see CWE-20). (Effectiveness: N/A)

**•** Requirements: Fully specify which encodings are required by components that will be communicating with each other. (Effectiveness: N/A)

**•** Implementation: When exchanging data between components, ensure that both components are using the same character encoding. Ensure that the proper encoding is applied at each interface. Explicitly set the encoding you are using whenever the protocol allows you to do so. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** The value read from the form parameter is reflected back to the client browser without having been encoded prior to output, allowing various XSS attacks (CWE-79).

**•** First let's look at the back end command processor code

**•** The programmer attempts to encode dangerous characters, however the denylist for encoding is incomplete (CWE-184) and an attacker can still pass a semicolon, resulting in a chain with command injection (CWE-77).

## Notes

**•** Relationship: This weakness is primary to all weaknesses related to injection (CWE-74) since the inherent nature of injection involves the violation of structured messages.

**•** Relationship: CWE-116 and CWE-20 have a close association because, depending on the nature of the structured message, proper input validation can indirectly prevent special characters from changing the meaning of a structured message. For example, by validating that a numeric ID field should only contain the 0-9 characters, the programmer effectively prevents injection attacks. However, input validation is not always sufficient, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a last name is inserted into a query. The name "O'Reilly" would likely pass the validation step since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or otherwise neutralized. In this case, stripping the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded.

**•** Terminology: The usage of the "encoding" and "escaping" terms varies widely. For example, in some programming languages, the terms are used interchangeably, while other languages provide APIs that use both terms for different tasks. This overlapping usage extends to the Web, such as the "escape" JavaScript function whose purpose is stated to be encoding. The concepts of encoding and escaping predate the Web by decades. Given such a context, it is difficult for CWE to adopt a consistent vocabulary that will not be misinterpreted by some constituency.

**•** Theoretical: This is a data/directive boundary error in which data boundaries are not sufficiently enforced before it is sent to a different control sphere.

**•** Research Gap: While many published vulnerabilities are related to insufficient output encoding, there is such an emphasis on input validation as a protection mechanism that the underlying causes are rarely described. Within CVE, the focus is primarily on well-understood issues like cross-site scripting and SQL injection. It is likely that this weakness frequently occurs in custom protocols that support multiple encodings, which are not necessarily detectable with automated techniques.