# CWE Detail – CWE-96

## Description

The product receives input from an upstream component, but it does not neutralize or incorrectly neutralizes code syntax before inserting the input into an executable resource, such as a library, configuration file, or template.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2002-0495: Perl code directly injected into CGI library file from parameters to another CGI program.

**•** CVE-2005-1876: Direct PHP code injection into supporting template file.

**•** CVE-2005-1894: Direct code injection into PHP script that can be accessed by attacker.

**•** CVE-2003-0395: PHP code from User-Agent HTTP header directly inserted into log file implemented as PHP script.

**•** CVE-2007-6652: chain: execution after redirect allows non-administrator to perform static code injection.

## Related Attack Patterns (CAPEC)

* CAPEC-35
* CAPEC-73
* CAPEC-77
* CAPEC-81
* CAPEC-85

## Attack TTPs

**•** T1027.009: Embedded Payloads (Tactics: defense-evasion)

**•** T1564.009: Resource Forking (Tactics: defense-evasion)

**•** T1027.006: HTML Smuggling (Tactics: defense-evasion)

## Modes of Introduction

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

**•** Implementation: This issue is frequently found in PHP applications that allow users to set configuration variables that are stored within executable PHP files. Technically, this could also be performed in some compiled code (e.g., by byte-patching an executable), although it is highly unlikely.

## Common Consequences

**•** Impact: Read Files or Directories, Read Application Data — Notes: The injected code could access restricted data / files.

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

**•** Impact: Gain Privileges or Assume Identity — Notes: Injected code can access resources that the attacker is directly prevented from accessing.

**•** Impact: Execute Unauthorized Code or Commands — Notes: Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing. Additionally, code injection can often result in the execution of arbitrary code.

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

## Potential Mitigations

**•** 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: Perform proper output validation and escaping to neutralize all code syntax from data written to code files. (Effectiveness: N/A)

## Applicable Platforms

**•** PHP (Class: None, Prevalence: Undetermined)

**•** Perl (Class: None, Prevalence: Undetermined)

**•** None (Class: Interpreted, Prevalence: Undetermined)

## Demonstrative Examples

**•** While the programmer intends for the MessageFile to only include data, an attacker can provide a message such as:

## Notes

**•** Relationship: "HTML injection" (see CWE-79: XSS) could be thought of as an example of this, but the code is injected and executed on the client side, not the server side. Server-Side Includes (SSI) are an example of direct static code injection.