# CWE Detail – CWE-211

## Description

The product performs an operation that triggers an external diagnostic or error message that is not directly generated or controlled by the product, such as an error generated by the programming language interpreter that a software application uses. The error can contain sensitive system information.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2004-1581: chain: product does not protect against direct request of an include file, leading to resultant path disclosure when the include file does not successfully execute.

**•** CVE-2004-1579: Single "'" inserted into SQL query leads to invalid SQL query execution, triggering full path disclosure. Possibly resultant from more general SQL injection issue.

**•** CVE-2005-0459: chain: product does not protect against direct request of a library file, leading to resultant path disclosure when the file does not successfully execute.

**•** CVE-2005-0443: invalid parameter triggers a failure to find an include file, leading to infoleak in error message.

**•** CVE-2005-0433: Various invalid requests lead to information leak in verbose error messages describing the failure to instantiate a class, open a configuration file, or execute an undefined function.

**•** CVE-2004-1101: Improper handling of filename request with trailing "/" causes multiple consequences, including information leak in Visual Basic error message.

## Modes of Introduction

**•** Architecture and Design: PHP applications are often targeted for having this issue when the PHP interpreter generates the error outside of the application's control. However, other languages/environments exhibit the same issue.

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

**•** Operation: N/A

## Common Consequences

**•** Impact: Read Application Data — Notes:

## Potential Mitigations

**•** System Configuration: Configure the application's environment in a way that prevents errors from being generated. For example, in PHP, disable display\_errors. (Effectiveness: N/A)

**•** Implementation: Debugging information should not make its way into a production release. (Effectiveness: N/A)

**•** Implementation: Debugging information should not make its way into a production release. (Effectiveness: N/A)

**•** Implementation: Handle exceptions internally and do not display errors containing potentially sensitive information to a user. Create default error pages if necessary. (Effectiveness: N/A)

**•** Implementation: The best way to prevent this weakness during implementation is to avoid any bugs that could trigger the external error message. This typically happens when the program encounters fatal errors, such as a divide-by-zero. You will not always be able to control the use of error pages, and you might not be using a language that handles exceptions. (Effectiveness: N/A)

## Applicable Platforms

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

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

## Demonstrative Examples

**•** N/A

**•** However, the error message output to the user contains information regarding the default directory on the local file system. This information can be exploited and may lead to unauthorized access or use of the system. Any Java RuntimeExceptions that are handled should not expose sensitive information to the user.

## Notes

**•** Relationship: This is inherently a resultant vulnerability from a weakness within the product or an interaction error.