# CWE Detail – CWE-98

## Description

The PHP application receives input from an upstream component, but it does not restrict or incorrectly restricts the input before its usage in "require," "include," or similar functions.

## Extended Description

In certain versions and configurations of PHP, this can allow an attacker to specify a URL to a remote location from which the product will obtain the code to execute. In other cases in association with path traversal, the attacker can specify a local file that may contain executable statements that can be parsed by PHP.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2004-0285: Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.

**•** CVE-2004-0030: Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.

**•** CVE-2004-0068: Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.

**•** CVE-2005-2157: Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.

**•** CVE-2005-2162: Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.

**•** CVE-2005-2198: Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.

**•** CVE-2004-0128: Modification of assumed-immutable variable in configuration script leads to file inclusion.

**•** CVE-2005-1864: PHP file inclusion.

**•** CVE-2005-1869: PHP file inclusion.

**•** CVE-2005-1870: PHP file inclusion.

**•** CVE-2005-2154: PHP local file inclusion.

**•** CVE-2002-1704: PHP remote file include.

**•** CVE-2002-1707: PHP remote file include.

**•** CVE-2005-1964: PHP remote file include.

**•** CVE-2005-1681: PHP remote file include.

**•** CVE-2005-2086: PHP remote file include.

**•** CVE-2004-0127: Directory traversal vulnerability in PHP include statement.

**•** CVE-2005-1971: Directory traversal vulnerability in PHP include statement.

**•** CVE-2005-3335: PHP file inclusion issue, both remote and local; local include uses ".." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector.

**•** CVE-2009-1936: chain: library file sends a redirect if it is directly requested but continues to execute, allowing remote file inclusion and path traversal.

## Related Attack Patterns (CAPEC)

* CAPEC-193

## Modes of Introduction

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

## Common Consequences

**•** Impact: Execute Unauthorized Code or Commands — Notes: The attacker may be able to specify arbitrary code to be executed from a remote location. Alternatively, it may be possible to use normal program behavior to insert php code into files on the local machine which can then be included and force the code to execute since php ignores everything in the file except for the content between php specifiers.

## 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. (Effectiveness: N/A)

**•** Architecture and Design: When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap [REF-185] provide this capability. (Effectiveness: N/A)

**•** Architecture and Design: For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. (Effectiveness: N/A)

**•** Architecture and Design: Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails. (Effectiveness: Limited)

**•** Architecture and Design: Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations. (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. When validating filenames, use stringent lists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../...//" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string. (Effectiveness: High)

**•** Architecture and Design: Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface. (Effectiveness: N/A)

**•** Architecture and Design: Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components. (Effectiveness: N/A)

**•** Operation: Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth. (Effectiveness: Moderate)

**•** Operation: Develop and run your code in the most recent versions of PHP available, preferably PHP 6 or later. Many of the highly risky features in earlier PHP interpreters have been removed, restricted, or disabled by default. (Effectiveness: N/A)

**•** Operation: When using PHP, configure the application so that it does not use register\_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register\_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues. Often, programmers do not protect direct access to files intended only to be included by core programs. These include files may assume that critical variables have already been initialized by the calling program. As a result, the use of register\_globals combined with the ability to directly access the include file may allow attackers to conduct file inclusion attacks. This remains an extremely common pattern as of 2009. (Effectiveness: N/A)

**•** Operation: Set allow\_url\_fopen to false, which limits the ability to include files from remote locations. (Effectiveness: High)

## Applicable Platforms

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

## Demonstrative Examples

**•** The problem with the above code is that the value of $dir is not restricted in any way, and a malicious user could manipulate the 'module\_name' parameter to force inclusion of an unanticipated file. For example, an attacker could request the above PHP page (example.php) with a 'module\_name' of "http://malicious.example.com" by using the following request string:

## Notes

**•** Relationship: This is frequently a functional consequence of other weaknesses. It is usually multi-factor with other factors (e.g. MAID), although not all inclusion bugs involve assumed-immutable data. Direct request weaknesses frequently play a role. Can overlap directory traversal in local inclusion problems.