# CWE Detail – CWE-73

## Description

The product allows user input to control or influence paths or file names that are used in filesystem operations.

## Extended Description

This could allow an attacker to access or modify system files or other files that are critical to the application. Path manipulation errors occur when the following two conditions are met: 1. An attacker can specify a path used in an operation on the filesystem. 2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted. For example, the program may give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2022-45918: Chain: a learning management tool debugger uses external input to locate previous session logs (CWE-73) and does not properly validate the given path (CWE-20), allowing for filesystem path traversal using "../" sequences (CWE-24)

**•** CVE-2008-5748: Chain: external control of values for user's desired language and theme enables path traversal.

**•** CVE-2008-5764: Chain: external control of user's target language enables remote file inclusion.

## Related Attack Patterns (CAPEC)

* CAPEC-13
* CAPEC-267
* CAPEC-64
* CAPEC-72
* CAPEC-76
* CAPEC-78
* CAPEC-79
* CAPEC-80

## Attack TTPs

**•** T1574.007: Path Interception by PATH Environment Variable (Tactics: persistence, privilege-escalation, defense-evasion)

**•** T1574.006: Dynamic Linker Hijacking (Tactics: persistence, privilege-escalation, defense-evasion)

**•** T1562.003: Impair Command History Logging (Tactics: defense-evasion)

**•** T1027: Obfuscated Files or Information (Tactics: defense-evasion)

## Modes of Introduction

**•** Architecture and Design: N/A

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

## Common Consequences

**•** Impact: Read Files or Directories, Modify Files or Directories — Notes: The application can operate on unexpected files. Confidentiality is violated when the targeted filename is not directly readable by the attacker.

**•** Impact: Modify Files or Directories, Execute Unauthorized Code or Commands — Notes: The application can operate on unexpected files. This may violate integrity if the filename is written to, or if the filename is for a program or other form of executable code.

**•** Impact: DoS: Crash, Exit, or Restart, DoS: Resource Consumption (Other) — Notes: The application can operate on unexpected files. Availability can be violated if the attacker specifies an unexpected file that the application modifies. Availability can also be affected if the attacker specifies a filename for a large file, or points to a special device or a file that does not have the format that the application expects.

## Potential Mitigations

**•** Architecture and Design: When the set of filenames is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames, 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 provide this capability. (Effectiveness: N/A)

**•** Architecture and Design: Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict all access to files within a particular directory. Examples include the Unix chroot jail and AppArmor. In general, managed code may provide some protection. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails. (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)

**•** 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 allowlists 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)

**•** Implementation: Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links (CWE-23, CWE-59). (Effectiveness: N/A)

**•** Installation: Use OS-level permissions and run as a low-privileged user to limit the scope of any successful attack. (Effectiveness: N/A)

**•** Operation: If you are using PHP, configure your application so that it does not use register\_globals. During implementation, develop your 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. (Effectiveness: N/A)

**•** Testing: Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** N/A

**•** N/A

## Notes

**•** Maintenance: CWE-114 is a Class, but it is listed a child of CWE-73 in view 1000. This suggests some abstraction problems that should be resolved in future versions.

**•** Relationship: The external control of filenames can be the primary link in chains with other file-related weaknesses, as seen in the CanPrecede relationships. This is because software systems use files for many different purposes: to execute programs, load code libraries, to store application data, to store configuration settings, record temporary data, act as signals or semaphores to other processes, etc. However, those weaknesses do not always require external control. For example, link-following weaknesses (CWE-59) often involve pathnames that are not controllable by the attacker at all. The external control can be resultant from other issues. For example, in PHP applications, the register\_globals setting can allow an attacker to modify variables that the programmer thought were immutable, enabling file inclusion (CWE-98) and path traversal (CWE-22). Operating with excessive privileges (CWE-250) might allow an attacker to specify an input filename that is not directly readable by the attacker, but is accessible to the privileged program. A buffer overflow (CWE-119) might give an attacker control over nearby memory locations that are related to pathnames, but were not directly modifiable by the attacker.