# CWE Detail – CWE-41

## Description

The product is vulnerable to file system contents disclosure through path equivalence. Path equivalence involves the use of special characters in file and directory names. The associated manipulations are intended to generate multiple names for the same object.

## Extended Description

Path equivalence is usually employed in order to circumvent access controls expressed using an incomplete set of file name or file path representations. This is different from path traversal, wherein the manipulations are performed to generate a name for a different object.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2000-1114: Source code disclosure using trailing dot

**•** CVE-2002-1986: Source code disclosure using trailing dot

**•** CVE-2004-2213: Source code disclosure using trailing dot or trailing encoding space "%20"

**•** CVE-2005-3293: Source code disclosure using trailing dot

**•** CVE-2004-0061: Bypass directory access restrictions using trailing dot in URL

**•** CVE-2000-1133: Bypass directory access restrictions using trailing dot in URL

**•** CVE-2001-1386: Bypass check for ".lnk" extension using ".lnk."

**•** CVE-2001-0693: Source disclosure via trailing encoded space "%20"

**•** CVE-2001-0778: Source disclosure via trailing encoded space "%20"

**•** CVE-2001-1248: Source disclosure via trailing encoded space "%20"

**•** CVE-2004-0280: Source disclosure via trailing encoded space "%20"

**•** CVE-2005-0622: Source disclosure via trailing encoded space "%20"

**•** CVE-2005-1656: Source disclosure via trailing encoded space "%20"

**•** CVE-2002-1603: Source disclosure via trailing encoded space "%20"

**•** CVE-2001-0054: Multi-Factor Vulnerability (MFV). directory traversal and other issues in FTP server using Web encodings such as "%20"; certain manipulations have unusual side effects.

**•** CVE-2002-1451: Trailing space ("+" in query string) leads to source code disclosure.

**•** CVE-2000-0293: Filenames with spaces allow arbitrary file deletion when the product does not properly quote them; some overlap with path traversal.

**•** CVE-2001-1567: "+" characters in query string converted to spaces before sensitive file/extension (internal space), leading to bypass of access restrictions to the file.

**•** CVE-2002-0253: Overlaps infoleak

**•** CVE-2001-0446: Application server allows remote attackers to read source code for .jsp files by appending a / to the requested URL.

**•** CVE-2004-0334: Bypass Basic Authentication for files using trailing "/"

**•** CVE-2001-0893: Read sensitive files with trailing "/"

**•** CVE-2001-0892: Web server allows remote attackers to view sensitive files under the document root (such as .htpasswd) via a GET request with a trailing /.

**•** CVE-2004-1814: Directory traversal vulnerability in server allows remote attackers to read protected files via .. (dot dot) sequences in an HTTP request.

**•** CVE-2002-1483: Read files with full pathname using multiple internal slash.

**•** CVE-1999-1456: Server allows remote attackers to read arbitrary files via a GET request with more than one leading / (slash) character in the filename.

**•** CVE-2004-0578: Server allows remote attackers to read arbitrary files via leading slash (//) characters in a URL request.

**•** CVE-2002-0275: Server allows remote attackers to bypass authentication and read restricted files via an extra / (slash) in the requested URL.

**•** CVE-2004-1032: Product allows local users to delete arbitrary files or create arbitrary empty files via a target filename with a large number of leading slash (/) characters.

**•** CVE-2002-1238: Server allows remote attackers to bypass access restrictions for files via an HTTP request with a sequence of multiple / (slash) characters such as http://www.example.com///file/.

**•** CVE-2004-1878: Product allows remote attackers to bypass authentication, obtain sensitive information, or gain access via a direct request to admin/user.pl preceded by // (double leading slash).

**•** CVE-2005-1365: Server allows remote attackers to execute arbitrary commands via a URL with multiple leading "/" (slash) characters and ".." sequences.

**•** CVE-2000-1050: Access directory using multiple leading slash.

**•** CVE-2001-1072: Bypass access restrictions via multiple leading slash, which causes a regular expression to fail.

**•** CVE-2004-0235: Archive extracts to arbitrary files using multiple leading slash in filenames in the archive.

**•** CVE-2002-1078: Directory listings in web server using multiple trailing slash

**•** CVE-2004-0847: ASP.NET allows remote attackers to bypass authentication for .aspx files in restricted directories via a request containing a (1) "\" (backslash) or (2) "%5C" (encoded backslash), aka "Path Validation Vulnerability."

**•** CVE-2000-0004: Server allows remote attackers to read source code for executable files by inserting a . (dot) into the URL.

**•** CVE-2002-0304: Server allows remote attackers to read password-protected files via a /./ in the HTTP request.

**•** CVE-1999-1083: Possibly (could be a cleansing error)

**•** CVE-2004-0815: "/./////etc" cleansed to ".///etc" then "/etc"

**•** CVE-2002-0112: Server allows remote attackers to view password protected files via /./ in the URL.

**•** CVE-2004-0696: List directories using desired path and "\*"

**•** CVE-2002-0433: List files in web server using "\*.ext"

**•** CVE-2001-1152: Proxy allows remote attackers to bypass denylist restrictions and connect to unauthorized web servers by modifying the requested URL, including (1) a // (double slash), (2) a /SUBDIR/.. where the desired file is in the parentdir, (3) a /./, or (4) URL-encoded characters.

**•** CVE-2000-0191: application check access for restricted URL before canonicalization

**•** CVE-2005-1366: CGI source disclosure using "dirname/../cgi-bin"

**•** CVE-1999-0012: Multiple web servers allow restriction bypass using 8.3 names instead of long names

**•** CVE-2001-0795: Source code disclosure using 8.3 file name.

**•** CVE-2005-0471: Multi-Factor Vulnerability. Product generates temporary filenames using long filenames, which become predictable in 8.3 format.

## Related Attack Patterns (CAPEC)

* CAPEC-3

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Read Files or Directories, Modify Files or Directories, Bypass Protection Mechanism — Notes: An attacker may be able to traverse the file system to unintended locations and read or overwrite the contents of unexpected files. If the files are used for a security mechanism than an attacker may be able to bypass the mechanism.

## 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: Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. (Effectiveness: N/A)

**•** Implementation: Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked. (Effectiveness: N/A)

## Applicable Platforms

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

## Notes

**•** Relationship: Some of these manipulations could be effective in path traversal issues, too.