# CWE Detail – CWE-39

## Description

The product accepts input that contains a drive letter or Windows volume letter ('C:dirname') that potentially redirects access to an unintended location or arbitrary file.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2001-0038: Remote attackers can read arbitrary files by specifying the drive letter in the requested URL.

**•** CVE-2001-0255: FTP server allows remote attackers to list arbitrary directories by using the "ls" command and including the drive letter name (e.g. C:) in the requested pathname.

**•** CVE-2001-0687: FTP server allows a remote attacker to retrieve privileged system information by specifying arbitrary paths.

**•** CVE-2001-0933: FTP server allows remote attackers to list the contents of arbitrary drives via a ls command that includes the drive letter as an argument.

**•** CVE-2002-0466: Server allows remote attackers to browse arbitrary directories via a full pathname in the arguments to certain dynamic pages.

**•** CVE-2002-1483: Remote attackers can read arbitrary files via an HTTP request whose argument is a filename of the form "C:" (Drive letter), "//absolute/path", or ".." .

**•** CVE-2004-2488: FTP server read/access arbitrary files using "C:\" filenames

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Execute Unauthorized Code or Commands — Notes: The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.

**•** Impact: Modify Files or Directories — Notes: The attacker may be able to overwrite or create critical files, such as programs, libraries, or important data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, appending a new account at the end of a password file may allow an attacker to bypass authentication.

**•** Impact: Read Files or Directories — Notes: The attacker may be able read the contents of unexpected files and expose sensitive data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, by reading a password file, the attacker could conduct brute force password guessing attacks in order to break into an account on the system.

**•** Impact: DoS: Crash, Exit, or Restart — Notes: The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the software from working at all and in the case of a protection mechanisms such as authentication, it has the potential to lockout every user of the software.

## 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. 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: 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)