# CWE Detail – CWE-426

## Description

The product searches for critical resources using an externally-supplied search path that can point to resources that are not under the product's direct control.

## Extended Description

This might allow attackers to execute their own programs, access unauthorized data files, or modify configuration in unexpected ways. If the product uses a search path to locate critical resources such as programs, then an attacker could modify that search path to point to a malicious program, which the targeted product would then execute. The problem extends to any type of critical resource that the product trusts. Some of the most common variants of untrusted search path are: In various UNIX and Linux-based systems, the PATH environment variable may be consulted to locate executable programs, and LD\_PRELOAD may be used to locate a separate library. In various Microsoft-based systems, the PATH environment variable is consulted to locate a DLL, if the DLL is not found in other paths that appear earlier in the search order.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-1999-1120: Application relies on its PATH environment variable to find and execute program.

**•** CVE-2008-1810: Database application relies on its PATH environment variable to find and execute program.

**•** CVE-2007-2027: Chain: untrusted search path enabling resultant format string by loading malicious internationalization messages.

**•** CVE-2008-3485: Untrusted search path using malicious .EXE in Windows environment.

**•** CVE-2008-2613: setuid program allows compromise using path that finds and loads a malicious library.

**•** CVE-2008-1319: Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded.

## Related Attack Patterns (CAPEC)

* CAPEC-38

## Attack TTPs

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

**•** T1574.009: Path Interception by Unquoted Path (Tactics: persistence, privilege-escalation, defense-evasion)

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Gain Privileges or Assume Identity, Execute Unauthorized Code or Commands — Notes: There is the potential for arbitrary code execution with privileges of the vulnerable program.

**•** Impact: DoS: Crash, Exit, or Restart — Notes: The program could be redirected to the wrong files, potentially triggering a crash or hang when the targeted file is too large or does not have the expected format.

**•** Impact: Read Files or Directories — Notes: The program could send the output of unauthorized files to the attacker.

## Potential Mitigations

**•** Architecture and Design: Hard-code the search path to a set of known-safe values (such as system directories), or only allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party. Be careful to avoid related weaknesses such as CWE-426 and CWE-428. (Effectiveness: N/A)

**•** Implementation: When invoking other programs, specify those programs using fully-qualified pathnames. While this is an effective approach, code that uses fully-qualified pathnames might not be portable to other systems that do not use the same pathnames. The portability can be improved by locating the full-qualified paths in a centralized, easily-modifiable location within the source code, and having the code refer to these paths. (Effectiveness: N/A)

**•** Implementation: Remove or restrict all environment settings before invoking other programs. This includes the PATH environment variable, LD\_LIBRARY\_PATH, and other settings that identify the location of code libraries, and any application-specific search paths. (Effectiveness: N/A)

**•** Implementation: Check your search path before use and remove any elements that are likely to be unsafe, such as the current working directory or a temporary files directory. (Effectiveness: N/A)

**•** Implementation: Use other functions that require explicit paths. Making use of any of the other readily available functions that require explicit paths is a safe way to avoid this problem. For example, system() in C does not require a full path since the shell can take care of it, while execl() and execv() require a full path. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** This code may look harmless at first, since both the directory and the command are set to fixed values that the attacker can't control. The attacker can only see the contents for DIR, which is the intended program behavior. Finally, the programmer is also careful to limit the code that executes with raised privileges.

**•** The code above allows an attacker to execute arbitrary commands with the elevated privilege of the application by modifying the system property APPHOME to point to a different path containing a malicious version of INITCMD. Because the program does not validate the value read from the environment, if an attacker can control the value of the system property APPHOME, then they can fool the application into running malicious code and take control of the system.

**•** If invoked by an unauthorized web user, it is providing a web page of potentially sensitive information on the underlying system, such as command-line arguments (CWE-497). This program is also potentially vulnerable to a PATH based attack (CWE-426), as an attacker may be able to create malicious versions of the ps or grep commands. While the program does not explicitly raise privileges to run the system commands, the PHP interpreter may by default be running with higher privileges than users.

**•** The problem here is that the program does not specify an absolute path for make and does not clean its environment prior to executing the call to Runtime.exec(). If an attacker can modify the $PATH variable to point to a malicious binary called make and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's make will now be run with these privileges, possibly giving the attacker complete control of the system.