# CWE Detail – CWE-642

## Description

The product stores security-critical state information about its users, or the product itself, in a location that is accessible to unauthorized actors.

## Extended Description

If an attacker can modify the state information without detection, then it could be used to perform unauthorized actions or access unexpected resources, since the application programmer does not expect that the state can be changed. State information can be stored in various locations such as a cookie, in a hidden web form field, input parameter or argument, an environment variable, a database record, within a settings file, etc. All of these locations have the potential to be modified by an attacker. When this state information is used to control security or determine resource usage, then it may create a vulnerability. For example, an application may perform authentication, then save the state in an "authenticated=true" cookie. An attacker may simply create this cookie in order to bypass the authentication.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2005-2428: Mail client stores password hashes for unrelated accounts in a hidden form field.

**•** CVE-2008-0306: Privileged program trusts user-specified environment variable to modify critical configuration settings.

**•** CVE-1999-0073: Telnet daemon allows remote clients to specify critical environment variables for the server, leading to code execution.

**•** CVE-2007-4432: Untrusted search path vulnerability through modified LD\_LIBRARY\_PATH environment variable.

**•** CVE-2006-7191: Untrusted search path vulnerability through modified LD\_LIBRARY\_PATH environment variable.

**•** CVE-2008-5738: Calendar application allows bypass of authentication by setting a certain cookie value to 1.

**•** CVE-2008-5642: Setting of a language preference in a cookie enables path traversal attack.

**•** CVE-2008-5125: Application allows admin privileges by setting a cookie value to "admin."

**•** CVE-2008-5065: Application allows admin privileges by setting a cookie value to "admin."

**•** CVE-2008-4752: Application allows admin privileges by setting a cookie value to "admin."

**•** CVE-2000-0102: Shopping cart allows price modification via hidden form field.

**•** CVE-2000-0253: Shopping cart allows price modification via hidden form field.

**•** 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-21
* CAPEC-31

## Attack TTPs

**•** T1539: Steal Web Session Cookie (Tactics: credential-access)

**•** T1528: Steal Application Access Token (Tactics: credential-access)

**•** T1134: Access Token Manipulation (Tactics: defense-evasion, privilege-escalation)

## Modes of Introduction

**•** Architecture and Design: OMISSION: This weakness is caused by missing a security tactic during the architecture and design phase.

**•** Implementation: N/A

## Common Consequences

**•** Impact: Bypass Protection Mechanism, Gain Privileges or Assume Identity — Notes: An attacker could potentially modify the state in malicious ways. If the state is related to the privileges or level of authentication that the user has, then state modification might allow the user to bypass authentication or elevate privileges.

**•** Impact: Read Application Data — Notes: The state variables may contain sensitive information that should not be known by the client.

**•** Impact: DoS: Crash, Exit, or Restart — Notes: By modifying state variables, the attacker could violate the application's expectations for the contents of the state, leading to a denial of service due to an unexpected error condition.

## Potential Mitigations

**•** Architecture and Design: Understand all the potential locations that are accessible to attackers. For example, some programmers assume that cookies and hidden form fields cannot be modified by an attacker, or they may not consider that environment variables can be modified before a privileged program is invoked. (Effectiveness: N/A)

**•** Architecture and Design: Store state information and sensitive data on the server side only. Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions. If information must be stored on the client, do not do so without encryption and integrity checking, or otherwise having a mechanism on the server side to catch tampering. Use a message authentication code (MAC) algorithm, such as Hash Message Authentication Code (HMAC) [REF-529]. Apply this against the state or sensitive data that has to be exposed, which can guarantee the integrity of the data - i.e., that the data has not been modified. Ensure that a strong hash function is used (CWE-328). (Effectiveness: N/A)

**•** Architecture and Design: Store state information on the server side only. Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions. (Effectiveness: N/A)

**•** 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. With a stateless protocol such as HTTP, use some frameworks can maintain the state for you. Examples include ASP.NET View State and the OWASP ESAPI Session Management feature. Be careful of language features that provide state support, since these might be provided as a convenience to the programmer and may not be considering security. (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)

**•** 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. (Effectiveness: N/A)

**•** Testing: Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible. (Effectiveness: N/A)

**•** Testing: Use dynamic tools and techniques that interact with the product using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The product's operation may slow down, but it should not become unstable, crash, or generate incorrect results. (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

**•** N/A

**•** 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 programmer takes care to call a specific "ls" program and sets the HOMEDIR to a fixed value. However, an attacker can use a command such as "ENV HOMEDIR /secret/directory" to specify an alternate directory, enabling a path traversal attack (CWE-22). At the same time, other attacks are enabled as well, such as OS command injection (CWE-78) by setting HOMEDIR to a value such as "/tmp; rm -rf /". In this case, the programmer never intends for HOMEDIR to be modified, so input validation for HOMEDIR is not the solution. A partial solution would be an allowlist that only allows the LANG variable to be specified in the ENV command. Alternately, assuming this is an authenticated user, the language could be stored in a local file so that no ENV command at all would be needed.