# CWE Detail – CWE-665

## Description

The product does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used.

## Extended Description

This can have security implications when the associated resource is expected to have certain properties or values, such as a variable that determines whether a user has been authenticated or not.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2001-1471: chain: an invalid value prevents a library file from being included, skipping initialization of key variables, leading to resultant eval injection.

**•** CVE-2008-3637: Improper error checking in protection mechanism produces an uninitialized variable, allowing security bypass and code execution.

**•** CVE-2008-4197: Use of uninitialized memory may allow code execution.

**•** CVE-2008-2934: Free of an uninitialized pointer leads to crash and possible code execution.

**•** CVE-2007-3749: OS kernel does not reset a port when starting a setuid program, allowing local users to access the port and gain privileges.

**•** CVE-2008-0063: Product does not clear memory contents when generating an error message, leading to information leak.

**•** CVE-2008-0062: Lack of initialization triggers NULL pointer dereference or double-free.

**•** CVE-2008-0081: Uninitialized variable leads to code execution in popular desktop application.

**•** CVE-2008-3688: chain: Uninitialized variable leads to infinite loop.

**•** CVE-2008-3475: chain: Improper initialization leads to memory corruption.

**•** CVE-2008-5021: Composite: race condition allows attacker to modify an object while it is still being initialized, causing software to access uninitialized memory.

**•** CVE-2005-1036: Chain: Bypass of access restrictions due to improper authorization (CWE-862) of a user results from an improperly initialized (CWE-909) I/O permission bitmap

**•** CVE-2008-3597: chain: game server can access player data structures before initialization has happened leading to NULL dereference

**•** CVE-2009-2692: chain: uninitialized function pointers can be dereferenced allowing code execution

**•** CVE-2009-0949: chain: improper initialization of memory can lead to NULL dereference

**•** CVE-2009-3620: chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference

## Related Attack Patterns (CAPEC)

* CAPEC-26
* CAPEC-29

## Modes of Introduction

**•** Implementation: This weakness can occur in code paths that are not well-tested, such as rare error conditions. This is because the use of uninitialized data would be noticed as a bug during frequently-used functionality.

**•** Operation: N/A

## Common Consequences

**•** Impact: Read Memory, Read Application Data — Notes: When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.

**•** Impact: Bypass Protection Mechanism — Notes: If security-critical decisions rely on a variable having a "0" or equivalent value, and the programming language performs this initialization on behalf of the programmer, then a bypass of security may occur.

**•** Impact: DoS: Crash, Exit, or Restart — Notes: The uninitialized data may contain values that cause program flow to change in ways that the programmer did not intend. For example, if an uninitialized variable is used as an array index in C, then its previous contents may produce an index that is outside the range of the array, possibly causing a crash or an exit in other environments.

## Potential Mitigations

**•** Requirements: Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, in Java, if the programmer does not explicitly initialize a variable, then the code could produce a compile-time error (if the variable is local) or automatically initialize the variable to the default value for the variable's type. In Perl, if explicit initialization is not performed, then a default value of undef is assigned, which is interpreted as 0, false, or an equivalent value depending on the context in which the variable is accessed. (Effectiveness: N/A)

**•** Architecture and Design: Identify all variables and data stores that receive information from external sources, and apply input validation to make sure that they are only initialized to expected values. (Effectiveness: N/A)

**•** Implementation: Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage. (Effectiveness: N/A)

**•** Implementation: Pay close attention to complex conditionals that affect initialization, since some conditions might not perform the initialization. (Effectiveness: N/A)

**•** Implementation: Avoid race conditions (CWE-362) during initialization routines. (Effectiveness: N/A)

**•** Build and Compilation: Run or compile your product with settings that generate warnings about uninitialized variables or data. (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)

## Applicable Platforms

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

## Demonstrative Examples

**•** N/A

**•** If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

**•** This might seem innocent enough, but str was not initialized, so it contains random memory. As a result, str[0] might not contain the null terminator, so the copy might start at an offset other than 0. The consequences can vary, depending on the underlying memory.