# CWE Detail – CWE-457

## Description

The code uses a variable that has not been initialized, leading to unpredictable or unintended results.

## Extended Description

In some languages such as C and C++, stack variables are not initialized by default. They generally contain junk data with the contents of stack memory before the function was invoked. An attacker can sometimes control or read these contents. In other languages or conditions, a variable that is not explicitly initialized can be given a default value that has security implications, depending on the logic of the program. The presence of an uninitialized variable can sometimes indicate a typographic error in the code.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2019-15900: Chain: sscanf() call is used to check if a username and group exists, but the return value of sscanf() call is not checked (CWE-252), causing an uninitialized variable to be checked (CWE-457), returning success to allow authorization bypass for executing a privileged (CWE-863).

**•** CVE-2008-3688: Chain: A denial of service may be caused by an uninitialized variable (CWE-457) allowing an infinite loop (CWE-835) resulting from a connection to an unresponsive server.

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

**•** CVE-2007-4682: Crafted input triggers dereference of an uninitialized object pointer.

**•** CVE-2007-3468: Crafted audio file triggers crash when an uninitialized variable is used.

**•** CVE-2007-2728: Uninitialized random seed variable used.

## Modes of Introduction

**•** Implementation: In C, using an uninitialized char \* in some string libraries will return incorrect results, as the libraries expect the null terminator to always be at the end of a string, even if the string is empty.

## Common Consequences

**•** Impact: Other — Notes: Initial variables usually contain junk, which can not be trusted for consistency. This can lead to denial of service conditions, or modify control flow in unexpected ways. In some cases, an attacker can "pre-initialize" the variable using previous actions, which might enable code execution. This can cause a race condition if a lock variable check passes when it should not.

**•** Impact: Other — Notes: Strings that are not initialized are especially dangerous, since many functions expect a null at the end -- and only at the end -- of a string.

## Potential Mitigations

**•** Implementation: Assign all variables to an initial value. (Effectiveness: N/A)

**•** Build and Compilation: Most compilers will complain about the use of uninitialized variables if warnings are turned on. (Effectiveness: N/A)

**•** Implementation: When using a language that does not require explicit declaration of variables, run or compile the software in a mode that reports undeclared or unknown variables. This may indicate the presence of a typographic error in the variable's name. (Effectiveness: N/A)

**•** Requirements: The choice could be made to use a language that is not susceptible to these issues. (Effectiveness: N/A)

**•** Architecture and Design: Mitigating technologies such as safe string libraries and container abstractions could be introduced. (Effectiveness: N/A)

## Applicable Platforms

**•** C (Class: None, Prevalence: Sometimes)

**•** C++ (Class: None, Prevalence: Sometimes)

**•** Perl (Class: None, Prevalence: Often)

**•** PHP (Class: None, Prevalence: Often)

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

## Demonstrative Examples

**•** This code checks if the POST array 'names' is set before assigning it to the $nameArray variable. However, if the array is not in the POST request, $nameArray will remain uninitialized. This will cause an error when the array is accessed to print the greeting message, which could lead to further exploit.

**•** In the default case of the switch statement, the programmer has accidentally set the value of aN twice. As a result, bN will have an undefined value. Most uninitialized variable issues result in general software reliability problems, but if attackers can intentionally trigger the use of an uninitialized variable, they might be able to launch a denial of service attack by crashing the program. Under the right circumstances, an attacker may be able to control the value of an uninitialized variable by affecting the values on the stack prior to the invocation of the function.

**•** When the printf() is reached,
 test\_string might be an unexpected address, so the
 printf might print junk strings (CWE-457). To fix this code, there are a couple approaches to
 making sure that test\_string has been properly set once
 it reaches the printf(). One solution would be to set test\_string to an
 acceptable default before the conditional: