# CWE Detail – CWE-170

## Description

The product does not terminate or incorrectly terminates a string or array with a null character or equivalent terminator.

## Extended Description

Null termination errors frequently occur in two different ways. An off-by-one error could cause a null to be written out of bounds, leading to an overflow. Or, a program could use a strncpy() function call incorrectly, which prevents a null terminator from being added at all. Other scenarios are possible.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2000-0312: Attacker does not null-terminate argv[] when invoking another program.

**•** CVE-2003-0777: Interrupted step causes resultant lack of null termination.

**•** CVE-2004-1072: Fault causes resultant lack of null termination, leading to buffer expansion.

**•** CVE-2001-1389: Multiple vulnerabilities related to improper null termination.

**•** CVE-2003-0143: Product does not null terminate a message buffer after snprintf-like call, leading to overflow.

**•** CVE-2009-2523: Chain: product does not handle when an input string is not NULL terminated (CWE-170), leading to buffer over-read (CWE-125) or heap-based buffer overflow (CWE-122).

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Read Memory, Execute Unauthorized Code or Commands — Notes: The case of an omitted null character is the most dangerous of the possible issues. This will almost certainly result in information disclosure, and possibly a buffer overflow condition, which may be exploited to execute arbitrary code.

**•** Impact: DoS: Crash, Exit, or Restart, Read Memory, DoS: Resource Consumption (CPU), DoS: Resource Consumption (Memory) — Notes: If a null character is omitted from a string, then most string-copying functions will read data until they locate a null character, even outside of the intended boundaries of the string. This could: cause a crash due to a segmentation fault cause sensitive adjacent memory to be copied and sent to an outsider trigger a buffer overflow when the copy is being written to a fixed-size buffer.

**•** Impact: Modify Memory, DoS: Crash, Exit, or Restart — Notes: Misplaced null characters may result in any number of security problems. The biggest issue is a subset of buffer overflow, and write-what-where conditions, where data corruption occurs from the writing of a null character over valid data, or even instructions. A randomly placed null character may put the system into an undefined state, and therefore make it prone to crashing. A misplaced null character may corrupt other data in memory.

**•** Impact: Alter Execution Logic, Execute Unauthorized Code or Commands — Notes: Should the null character corrupt the process flow, or affect a flag controlling access, it may lead to logical errors which allow for the execution of arbitrary code.

## Potential Mitigations

**•** Requirements: Use a language that is not susceptible to these issues. However, be careful of null byte interaction errors (CWE-626) with lower-level constructs that may be written in a language that is susceptible. (Effectiveness: N/A)

**•** Implementation: Ensure that all string functions used are understood fully as to how they append null characters. Also, be wary of off-by-one errors when appending nulls to the end of strings. (Effectiveness: N/A)

**•** Implementation: If performance constraints permit, special code can be added that validates null-termination of string buffers, this is a rather naive and error-prone solution. (Effectiveness: N/A)

**•** Implementation: Switch to bounded string manipulation functions. Inspect buffer lengths involved in the buffer overrun trace reported with the defect. (Effectiveness: N/A)

**•** Implementation: Add code that fills buffers with nulls (however, the length of buffers still needs to be inspected, to ensure that the non null-terminated string is not written at the physical end of the buffer). (Effectiveness: N/A)

## Applicable Platforms

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

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

## Demonstrative Examples

**•** The code above will behave correctly if the data read from cfgfile is null terminated on disk as expected. But if an attacker is able to modify this input so that it does not contain the expected NULL character, the call to strcpy() will continue copying from memory until it encounters an arbitrary NULL character. This will likely overflow the destination buffer and, if the attacker can control the contents of memory immediately following inputbuf, can leave the application susceptible to a buffer overflow attack.

**•** The code above will not always behave correctly as readlink() does not append a NULL byte to buf. Readlink() will stop copying characters once the maximum size of buf has been reached to avoid overflowing the buffer, this will leave the value buf not NULL terminated. In this situation, strlen() will continue traversing memory until it encounters an arbitrary NULL character further on down the stack, resulting in a length value that is much larger than the size of string. Readlink() does return the number of bytes copied, but when this return value is the same as stated buf size (in this case MAXPATH), it is impossible to know whether the pathname is precisely that many bytes long, or whether readlink() has truncated the name to avoid overrunning the buffer. In testing, vulnerabilities like this one might not be caught because the unused contents of buf and the memory immediately following it may be NULL, thereby causing strlen() to appear as if it is behaving correctly.

**•** The above code gives the following output: "The last character in shortString is: n (6e)". So, the shortString array does not end in a NULL character, even though the "safe" string function strncpy() was used. The reason is that strncpy() does not impliciitly add a NULL character at the end of the string when the source is equal in length or longer than the provided size.

## Notes

**•** Relationship: Factors: this is usually resultant from other weaknesses such as off-by-one errors, but it can be primary to boundary condition violations such as buffer overflows. In buffer overflows, it can act as an expander for assumed-immutable data.

**•** Relationship: Overlaps missing input terminator.

**•** Applicable Platform: Conceptually, this does not just apply to the C language; any language or representation that involves a terminator could have this type of problem.

**•** Maintenance: As currently described, this entry is more like a category than a weakness.