# CWE Detail – CWE-134

## Description

The product uses a function that accepts a format string as an argument, but the format string originates from an external source.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2002-1825: format string in Perl program

**•** CVE-2001-0717: format string in bad call to syslog function

**•** CVE-2002-0573: format string in bad call to syslog function

**•** CVE-2002-1788: format strings in NNTP server responses

**•** CVE-2006-2480: Format string vulnerability exploited by triggering errors or warnings, as demonstrated via format string specifiers in a .bmp filename.

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

## Related Attack Patterns (CAPEC)

* CAPEC-135
* CAPEC-67

## Modes of Introduction

**•** Implementation: The programmer rarely intends for a format string to be externally-controlled at all. This weakness is frequently introduced in code that constructs log messages, where a constant format string is omitted.

**•** Implementation: In cases such as localization and internationalization, the language-specific message repositories could be an avenue for exploitation, but the format string issue would be resultant, since attacker control of those repositories would also allow modification of message length, format, and content.

## Common Consequences

**•** Impact: Read Memory — Notes: Format string problems allow for information disclosure which can severely simplify exploitation of the program.

**•** Impact: Modify Memory, Execute Unauthorized Code or Commands — Notes: Format string problems can result in the execution of arbitrary code, buffer overflows, denial of service, or incorrect data representation.

## Potential Mitigations

**•** Requirements: Choose a language that is not subject to this flaw. (Effectiveness: N/A)

**•** Implementation: Ensure that all format string functions are passed a static string which cannot be controlled by the user, and that the proper number of arguments are always sent to that function as well. If at all possible, use functions that do not support the %n operator in format strings. [REF-116] [REF-117] (Effectiveness: N/A)

**•** Build and Compilation: Run compilers and linkers with high warning levels, since they may detect incorrect usage. (Effectiveness: N/A)

## Applicable Platforms

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

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

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

## Demonstrative Examples

**•** The example is exploitable, because of the call to printf() in the printWrapper() function. Note: The stack buffer was added to make exploitation more simple.

**•** This code allows an attacker to view the contents of the stack and write to the stack using a command line argument containing a sequence of formatting directives. The attacker can read from the stack by providing more formatting directives, such as %x, than the function takes as arguments to be formatted. (In this example, the function takes no arguments to be formatted.) By using the %n formatting directive, the attacker can write to the stack, causing snprintf() to write the number of bytes output thus far to the specified argument (rather than reading a value from the argument, which is the intended behavior). A sophisticated version of this attack will use four staggered writes to completely control the value of a pointer on the stack.

**•** This code produces the following output: 5 9 5 5 It is also possible to use half-writes (%hn) to accurately control arbitrary DWORDS in memory, which greatly reduces the complexity needed to execute an attack that would otherwise require four staggered writes, such as the one mentioned in a separate example.

## Notes

**•** Applicable Platform: This weakness is possible in any programming language that support format strings.

**•** Other: In some circumstances, such as internationalization, the set of format strings is externally controlled by design. If the source of these format strings is trusted (e.g. only contained in library files that are only modifiable by the system administrator), then the external control might not itself pose a vulnerability. While Format String vulnerabilities typically fall under the Buffer Overflow category, technically they are not overflowed buffers. The Format String vulnerability is fairly new (circa 1999) and stems from the fact that there is no realistic way for a function that takes a variable number of arguments to determine just how many arguments were passed in. The most common functions that take a variable number of arguments, including C-runtime functions, are the printf() family of calls. The Format String problem appears in a number of ways. A \*printf() call without a format specifier is dangerous and can be exploited. For example, printf(input); is exploitable, while printf(y, input); is not exploitable in that context. The result of the first call, used incorrectly, allows for an attacker to be able to peek at stack memory since the input string will be used as the format specifier. The attacker can stuff the input string with format specifiers and begin reading stack values, since the remaining parameters will be pulled from the stack. Worst case, this improper use may give away enough control to allow an arbitrary value (or values in the case of an exploit program) to be written into the memory of the running program. Frequently targeted entities are file names, process names, identifiers. Format string problems are a classic C/C++ issue that are now rare due to the ease of discovery. One main reason format string vulnerabilities can be exploited is due to the %n operator. The %n operator will write the number of characters, which have been printed by the format string therefore far, to the memory pointed to by its argument. Through skilled creation of a format string, a malicious user may use values on the stack to create a write-what-where condition. Once this is achieved, they can execute arbitrary code. Other operators can be used as well; for example, a %9999s operator could also trigger a buffer overflow, or when used in file-formatting functions like fprintf, it can generate a much larger output than intended.

**•** Research Gap: Format string issues are under-studied for languages other than C. Memory or disk consumption, control flow or variable alteration, and data corruption may result from format string exploitation in applications written in other languages such as Perl, PHP, Python, etc.