6.64 Uncontrolled format string [SHL]

6.64.1 Description of application vulnerability

Many languages use format string to control how output is generated or input acquired. If the contents of the format string can be influenced by external data, there is an opportunity for an attacker to gain access to what should be private data, to execute arbitrary code, or to cause resource exhaustion or buffer overrun. Even without an attacker, mistakes in format strings may cause serious program errors.

6.64.2 Cross reference

CWE:

134. Uncontrolled Format String

6.64.3 Mechanism of failure

Format strings are parameters of input or output functions. They consist of fixed text and control sequences that are associated with other parameters of the function, and which control how the parameters are displayed or loaded.

There are a number of mechanisms relating to format strings that can lead to safety and security problems.

Firstly, for an output function, the format string controls what is written to an output channel (file or printer) or a character buffer. In the latter case particularly there is the possibility of buffer overrun, when the format string causes data to be written beyond the end of the buffer. In most languages that provide I/O control using format strings, it is possible for control sequences in the format string to control the size of the value written (e.g. the control sequence `%6d` in C based languages means write an integer value in a 6 character field, padding with spaces if necessary). If the size of the target field is accidentally or maliciously increased (say to `%6000d`) then buffer overrun or resource exhaustion can easily occur.

As the format string controls what is written to an output channel, if an attacker can influence the format string, then they can control what is written to a buffer, which could include executable code. If the attacker can then cause corruption of the program stack, it may be possible to execute this code.

As the format string is interpreted at run-time and expects to find a parameter for each control sequence, if the format string has more control sequences than supplied parameters, it is likely that additional values will be read off the stack. This can lead to unexpected values being output, with the possibility of leakage of sensitive information.

A fourth, less common, vulnerability relates to the format string potentially being able to modify a data values passed for output. Again using C-based languages as an example, the `%n` control sequence means write the number of characters output so far by this function to the value pointed to by the associated parameter. If the function should be writing the value of an object who's address was supplied by a pointer, then if the intended control sequence is modified to `%n`, that value will be changed instead.

The programmer rarely intends for a format string to be user-controlled. However, this weakness frequently occurs in code that reads log messages from a file (for internationalization or user customization). Such messages may safely be output using a format string that is interpreted as 'output a string', but it is not unknown for the programmer to omit the format string and use the message to be output as the format string, expecting it to consist solely of literal text. If the message has been corrupted, so that it includes control sequences, any of the issues mentioned above may occur.

Delete
Delete
Delete
Delete

6.64.4 Applicable language characteristics

Delete
interna
reposit
format
of thos
messag

This vulnerability is intended to be applicable to languages with the following characteristics:

- Languages that support format strings for input/output functions.

6.64.5 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Ensure that all format string functions are passed as static string which cannot be controlled by the user and that the proper number of arguments is always sent to that function.
- Ensure all specifiers used match the associated parameter.
- Avoid format strings that will write to a memory location that is pointed to by its argument.
- Where a function expects a format string, always supply one, even if it is the apparently redundant 'write a string'. Don't use a non-static text string to be output as the format string.

Dele
Dele
Dele

6.64.6 Implications for language design and evolution

In future language design and evolution activities, the following items should be considered:

- Ensure all format strings are verified to be correct in regard to the associated argument or parameter.