Taint Analysis

Material from this section was contributed to ISO/IEC TS 17961:2013.

Taint and Tainted Sources

Certain operations and functions have a domain that is a subset of the type domain of their operands or parameters. When the actual values are outside of the defined domain, the result might be undefined or at least unexpected. If the value of an operand or argument may be outside the domain of an operation or function that consumes that value, and the value is derived from any external input to the program (such as a command-line argument, data returned from a system call, or data in shared memory), that value is tainted, and its origin is known as a tainted source. A tainted value is not necessarily known to be out of the domain; rather, it is not known to be in the domain. Only values, and not the operands or arguments, can be tainted; in some cases, the same operand or argument can hold tainted or untainted values along different paths. In this regard, taint is an attribute of a value that is assigned to any value originating from a tainted source.

Restricted Sinks

Operands and arguments whose domain is a subset of the domain described by their types are called restricted sinks. Any integer operand used in a pointer arithmetic operation is a restricted sink for that operand. Certain parameters of certain library functions are restricted sinks because these functions perform address arithmetic with these parameters, or control the allocation of a resource, or pass these parameters on to another restricted sink. All string input parameters to library functions are restricted sinks because it is possible to pass in a character sequence that is not null terminated. The exceptions are input parameters to `strncpy()` and `strncpy_s()`, which explicitly allow the source character sequence not to be null terminated.

Propagation

Taint is propagated through operations from operands to results unless the operation itself imposes constraints on the value of its result that subsume the constraints imposed by restricted sinks. In addition to operations that propagate the same sort of taint, there are operations that propagate taint of one sort of an operand to taint of a different sort for their results, the most notable example of which is `strlen()` propagating the taint of its argument with respect to string length to the taint of its return value with respect to range. Although the exit condition of a loop is not normally considered to be a restricted sink, a loop whose exit condition depends on a tainted value propagates taint to any numeric or pointer variables that are increased or decreased by amounts proportional to the number of iterations of the loop.

Sanitization

To remove the taint from a value, the value must be sanitized to ensure that it is in the defined domain of any restricted sink into which it flows. Sanitization is performed by replacement or termination. In replacement, out-of-domain values are replaced by in-domain values, and processing continues using an in-domain value in place of the original. In termination, the program logic terminates the path of execution when an out-of-domain value is detected, often simply by branching around whatever code would have used the value.

In general, sanitization cannot be recognized exactly using static analysis. Analyzers that perform taint analysis usually provide some extralinguistic mechanism to identify sanitizing functions that sanitize an argument (passed by address) in place, return a sanitized version of an argument, or return a status code indicating whether the argument is in the required domain. Because such extralinguistic mechanisms are outside the scope of this coding standard, we use a set of rudimentary definitions of sanitization that is likely to recognize real sanitization but might cause nonsanitizing or ineffectively sanitizing code to be misconstrued as sanitizing.

The following definition of sanitization presupposes that the analysis is in some way maintaining a set of constraints on each value encountered as the simulated execution progresses: a given path through the code sanitizes a value with respect to a given restricted sink if it restricts the range of that value to a subset of the defined domain of the restricted sink type. For example, sanitization of signed integers with respect to an array index operation must restrict the range of that integer value to numbers between zero and the size of the array minus one.

This description is suitable for numeric values, but sanitization of strings with respect to content is more difficult to recognize in a general way.