6.X Object method qualification [???]

6.X.1 Description of application vulnerability

A number of OO languages support the idea of member function qualification: for example in C++ const and static, in Ada *<what's the Ada – or other language's - equivalent>*. Where:

- a const member function doesn't alter (write) any member variables of a class instance, nor calls any non-const functions of the class
- a static member function doesn't access (read or write) any member variables of the class instance, nor calls any non-static functions of the class. A static function may access variables for which there is only one instance accessed by all objects of that class (in C++ terms, a static member variable)

It is generally regarded as good programming practice to document the intended use to be made of a variable when it is passed as a function parameter, by making the parameter const if there is no intension to modify the variable or non-const if it may be modified. This is particularly significant if the parameter is passed by reference rather than value.

If the parameter is of class type, and a call of a class member function is required, if the class member function is not qualified as const, the parameter must be non-const, as calling the function may modify the parameter object. Conversely, for a const parameter object, only const member function may be called on it.

6.X.2 Cross references

MISRA C++:2008 rule 9-3-3
JSF++ rule 69
+TBD

6.X.3 Mechanism of failure

If a class member function that could be const is not qualified as const, then the expected use of a function parameter of this class type (or reference) cannot be documented as const if that unqualified member function needs to be called.

Whilst not a failure as such, a valuable piece of documentation of programmer intent may be lost , which can impact future maintenance, with what was intended to be a constant object getting modified.

Similarly, if a class member function that could be static is not qualified as static, then that function will not be available to be used without access to an object of the required class type.

These two properties have to be considered together as they both consider what use is to be made of member variables of a class instance

6.X.4 Application language characteristics

This vulnerability applies to OO languages which support the notions of const and/or static member function qualification

6.X.5 Avoiding the vulnerability or mitigating its effects

A number of static analysis standards have rules along the lines 'any class member function that can be made static, shall be made static' 'any non-static class member function that can be made const shall be made const'

6.X.6 Implications for language design and evolution

Whether a class member function may be const or static is an easy property for a compiler to determine, As a minimum, compilers should give a warning if:

- a class member function that could be static is not qualified as static
- a class member function that could be const is not qualified as const