ISO/IEC JTC 1/SC 22/OWGV N 0190

Proposed rewrite of Sub-clause 6.26

Date	5 May 2009
Contributed by	Robert Karlin
Original file name	6-26 rewrite.doc
Notes	Closes Action Item #10-06

6.26 Inheritance [RIP]

6.26.1 Description of application vulnerability

Inheritance, the ability to create enhanced and/or restricted object classes based on existing object classes can introduce a number of vulnerabilities, both inadvertent and malicious. Because Inheritance allows the overriding of methods of the parent class and because object oriented systems are designed to separate and encapsulate code and data, it can be difficult to determine where in the hierarchy an invoked method is actually defined. Also, since an overriding method does not need to call the method in the parent class that has been overridden, essential initialization and manipulation of class data may be bypassed. This can be especially dangerous during constructor and destructor methods.

Languages that allow multiple inheritance add additional complexities to the resolution of method invocations. Different object brokerage systems may resolve the method identity to different classes, based on how the inheritance tree is traversed.

6.26.2 Cross reference

JSF AV Rules: 86 to 97
MISRA C++ 2008: 0-1-12, 8-3-1, 10-1-1 to 10-1-3, and 10-3-1 to 10-3-3

6.26.3 Mechanism of failure

The use of inheritance can lead to an exploitable application vulnerability or negatively impact system safety in a number of ways:

- Execution of malicious redefinitions - this can occur through the insertion of a class into the class hierarchy that overrides commonly called methods in the parent classes.
- Accidental redefinition - where a method is defined that inadvertently overrides a method that has already been defined in a parent class
- Accidental failure of redefinition - when a method is incorrectly named, or the parameters are not defined properly, and thus does not override a method in a parent class
- Breaking of class invariants - this can be cause by redefining methods that initialize or validate class data without including that initialization or validation in the overriding methods

These vulnerabilities can increase dramatically as the complexity of the hierarchy increases, especially in the use of multiple inheritance.

6.26.4 Applicable language characteristics

This is applicable to all languages that allow single and multiple inheritances.

6.26.5 Avoiding the vulnerability or mitigating its effects

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

- Avoid the use of multiple inheritance whenever possible
- Provide complete documentation of all encapsulated data, and how each method affects that data for each object in the hierarchy
- Inherit only from trusted sources, and, whenever possible, check the version of the parent classes during compilation and/or initialization
- Provide a method that provides versioning information for each class.

6.26.6 Implications for standardization

- Language specification should include the definition of a common versioning method
- Compilers should provide an option to report the class in which a resolved method resides
- Runtime environments should provide a trace of all runtime method resolutions

6.26.7 Bibliography

[1] P. V. Bhansali, A systematic approach to identifying a safe subset for safety-critical software, ACM SIGSOFT
Software Engineering Notes, v.28 n.4, July 2003
[2] Ghassan, A., & Alkadi, I. (2003). Application of a Revised DIT Metric to Redesign an OO Design. *Journal of*
Object Technology , 127-134.
[3] Subramanian, S., Tsai, W.-T., & Rayadurgam, S. (1998). Design Contraint Violation Detection in Safety-Critical
Systems. The 3rd IEEE International Symposium on High-Assurance Systems Engineering , 109 - 116.