John Munguia

CS 405

August 3, 2025

**Static Code Analysis Process Summary**

This analysis compares the results of Visual Studio's built-in compiler warnings and the Cppcheck static code analysis tool. The objective was to identify coding issues in a C++ source file provided for a simulated banking web application. The file was compiled in Visual Studio to capture syntax and warning messages, and then analyzed with Cppcheck version 2.18. The Cppcheck settings were configured to show error IDs, enable inline suppressions, check for inconclusive results, and enforce the C++17 standard. This summary documents issues found only by Cppcheck, their associated risk levels, and a brief explanation of each.

Overall, Cppcheck identified a wide range of issues, including logic errors, memory problems, and style violations, that Visual Studio did not report. While Visual Studio remains useful for detecting immediate syntax and type issues, Cppcheck provided deeper insight into runtime risks, dangerous patterns, and maintainability problems. Using both tools together offers the most comprehensive coverage of potential code problems.

1. **Assert Assignment**  
   Risk: High Risk  
   Found by: CppCheck  
   Cppcheck identified that variable z is modified within an assert statement. Since assertions are removed in release builds, the assignment will not execute in production, potentially causing incorrect logic during runtime.
2. **Assert Function Call**  
   Risk: High Risk  
   Found by: CppCheck  
   Cppcheck flagged that the function my\_function() is called within an assert statement. If the function performs important logic, it will be skipped in release builds, leading to unintended behavior.
3. **Pointer to Local Variable**  
   Risk: High Risk  
   Found by: CppCheck  
   In the foo function, a pointer is set to the address of a local variable b. Once the function exits, the pointer becomes invalid and may lead to a crash if accessed.
4. **Out-of-Bounds Access**  
   Risk: High Risk  
   Found by: CppCheck  
   Cppcheck found that the program writes to buf[1000], even though buf is only 10 elements in size. This causes an out-of-bounds write, which can corrupt memory or crash the application.
5. **Noexcept Violation**  
   Risk: High Risk  
   Found by: CppCheck  
   The function DontThrow() is marked noexcept but throws an exception. This violates the exception specification and can cause unexpected termination at runtime.
6. **Uninitialized Member**  
   Risk: Medium Risk  
   Found by: CppCheck  
   The copy constructor in class A does not initialize the x member variable. This leaves x with an undefined value, which could lead to unpredictable behavior when the object is used.
7. **Invalid Iterator Use**  
   Risk: High Risk  
   Found by: CppCheck  
   In vector\_test, the code erases an element from a vector while iterating through it. This invalidates the iterator and can result in crashes or incorrect behavior if not handled carefully.
8. **Null Dereference Risk**  
   Risk: High Risk  
   Found by: CppCheck  
   Cppcheck detected a possible null pointer dereference involving tok. The tool noted that either the null check is unnecessary or dereferencing tok may still occur when it is null, leading to a crash.
9. **Invalid Bool Comparison**  
   Risk: Medium Risk  
   Found by: CppCheck  
   A boolean value returned from my\_function() is compared to the integer 3. This condition will always be false and suggests faulty logic in the condition.
10. **Wrong Return Type**  
    Risk: Low Risk  
    Found by: CppCheck  
    The function my\_function() returns an integer instead of a true boolean. While this compiles, it introduces ambiguity and is a bad practice that can mislead others reviewing the code.
11. **Unused Functions**  
    Risk: Low Risk  
    Found by: CppCheck  
    Cppcheck found unused functions like DontThrow, do\_something\_useless, and is\_type. These functions do not affect execution but clutter the code and may confuse future developers.
12. **Shadowed Variables**  
    Risk: Low Risk  
    Found by: CppCheck  
    Variables x, y, and z inside the try block shadow global variables of the same name. This is legal but confusing, and it can cause subtle bugs during debugging or maintenance.
13. **Unused Assignments**  
    Risk: Low Risk  
    Found by: CppCheck  
    Variables such as tok, buf[count], x, and y are assigned values but never used. These might be leftovers from previous development and can be safely removed to clean up the code.
14. **Missing Static Keyword**  
    Risk: Low Risk  
    Found by: CppCheck  
    Functions DontThrow and Token::next can be marked as static because they do not rely on instance data. Making them static can slightly improve performance and code clarity.
15. **Variable Scope Suggestion**  
    Risk: Low Risk  
    Found by: CppCheck  
    Cppcheck suggested that the scope of the variable buf could be reduced. This change is optional, but it helps prevent unintended usage outside its needed context.

Overall, Cppcheck identified a wide range of issues, including logic errors, memory problems, and style violations, that Visual Studio did not report. While Visual Studio remains useful for detecting immediate syntax and type issues, Cppcheck provided deeper insight into runtime risks, dangerous patterns, and maintainability problems. Using both tools together offers the most comprehensive coverage of potential code problems.