5-3 Activity: Static Code Analysis

Joel Meza

Professor Trevor Hodde

February 3, 2022

CS 405 – Secure Coding

Southern New Hampshire University

**Overview Process Summary**

Overall, I believe that Visual Studios IDE can throw exceptions when bugs are to be allocated in an uncompliant source code. Although, I learned in this activity that C++ check is a great extension software tool for VS IDE as it helps to specialize on thrown exceptions and another set of eyes to detect errors being compiled before even running the code. This experience has shown me to take secure coding to a different level of security as it should be required when working in a software system or project. C++ check (CppCheck) is defined as a static depth analysis tool for C/C++ code. It provides unique code depth analysis to detect bugs and focuses on detecting undefined behavior and dangerous coding constructs. The goal is to have very few false positives thrown exceptions. C++ check is designed to be able to analyze your C/C++ code even if it has non-standard syntax (common in embedded projects).

Therefore, I experienced the difference between both the Visual Studios IDE alone and C++ check extension tool, on what had offered to me on compiled errors as shown below.

There were several errors found below which were matched by **Visual Studios** like:

1. Verified as Not Risk: Variable ‘tok’ is assigned a value that has never been declared.
2. Verified as Risk: Assert statements are removed from release builds so the code from inside is asserted to be unexecuted. Although, if the static code analysis is required while is in build mode, then it’s a bug.
3. Verified as Risk: Either condition count is equal to 1000, then it’s known to be redundant while the array ‘buf[10]’ has be pointed to an index of 1000 where it’s out of range.
4. Verified as Risk: The member instance variable “A::x” is not initialized as a intent constructor.

There were several errors found below which were matched by **CPP Check** like:

1. Verified as Risk: function parameter is assigned the address of a local auto-variable. Local auto-variables are isolated from the stack which is freed when the function ends.

(However, the pointer to a local variable is invalid after the function comes to an end.)

1. Verified as Not Risk: Variable ‘tok’ is assigned a value that has never been declared.
2. Verified as Risk: Assert statements are removed from release builds so the code from inside is asserted to be unexecuted.
3. Verified as Not Risk: The member instance variable “MySpecialType::DontThrow’ can be made a static function. Creating a statical function could give a performance, which “this” instance is pointed to the function. This modification wouldn’t cause a compiler error, but it does not require to be uncertain.
4. Verified as Not Risk: Local variable to be for [‘x’ ‘y’ ‘z’] to track the global variable.
5. Verified as Risk: Non-boolean value is returned due to function function not included bool. Otherwise, it shows that it has not included the library used for a boolean which needs to have in the head of the module as #include <stdbool.h>.
6. Verified as Risk: Boolean expression is in comparison to an integer other from zero to one.
7. Verified as Risk: Scope of a constant variable like “buf” that could be condensed. Warning: Be careful when fixing this message, consider any local loops.
8. Verified as Not Risk: Whether variable ‘tok’ considered to be redundant or could it be that a null pointer is not assigned to ‘tok’.
9. Verified as Risk: Using iterators to a local container like “items” is known to be invalid.
10. Verified as Not Risk: Expection\_Thrown: no function is declared to throw exceptions.
11. Verified as Risk: Function argument/parameter has no global effect to function.

Graphical user interface, text

Description automatically generated

Text

Description automatically generated