Larry Arndt

CS-405

February 3, 2023

Questionable.cpp Errors

Visual Basic:

Warning C6386: @ Line 64 => int buf[10];

This warning indicates that the writable extent of the specified buffer might be smaller than the index used to write to it. This defect can cause buffer overrun.

Warning C26495: @ Line 42 => A(const A& other) {}

A member variable isn't initialized by a constructor or by an initializer. Make sure all variables are initialized by the end of construction.

ccpCheck:

CWE: 398: @ Line 127 => assert(z = 2); @ Line 129 => assert(my\_function() == 3);

Line 127: (Variable 'z' is modified inside assert statement.) Line 129: (Non-pure function: 'my\_function' is called inside assert statement.) Assert statements are removed from release builds so the code inside assert statement is not executed. If the code is needed also in release builds, this is a bug.

CWE: 398 & CWE: 476 @ Line 109 => tok = tok->next();

398: Assignment of function parameter has no effect outside the function. It was not dereferenced. 476: Either the condition 'tok' is redundant or there is possible null pointer dereference: tok.

CWE: 562 @ Line 59 => \*a = &b;

Dangerous assignment - the function parameter is assigned the address of a local auto-variable. Local auto-variables are reserved from the stack which is freed when the function ends. So, the pointer to a local variable is invalid after the function ends.

CWE: 788: @ Line 66 => buf[count] = 0;

Either the condition 'count==1000' is redundant or the array 'buf[10]' is accessed at index 1000, which is out of bounds.

CWE: 398: @ Line 42 => A(const A& other) {}

Member variable 'A::x' is not initialized in the copy constructor.

CWE: 664: @ Line 87 => for (iter = items.begin(); iter != items.end(); ++iter) {

Using iterator to local container 'items' that may be invalid.

Source Code:

// QuestionableCode.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include <cassert>

#include <iostream>

#include <numeric>

#include <set>

#include <vector>

// TODO: You are going to compare the warnings & errors between Visual Studio and CppCheck.

// Make sure you are using CppCheck 2.1 or greater

//

// 1. Create a Visual Studio Console project with this file as the only file

// 2. Compile it and check the warning and errors

// 3. Create a CppCheck project to analyze a Visual Studio project with this file.

// 4. In CppCheck Edit / Preferences / General Tab: Set Display error id in column "Id", Enable inline suppressions, and Check for inconclusive errors also

// 5. In CppCheck View, Select Check All to ensure all types of checks are enabled

// 6. In CppCheck Analyze, set the C++ Standard to C++17, and Enforce C++

// 7. Make sure to run the analysis

// 8. Save the results to a file (XML format)

// 9. Take a screen shot of the Visual Studio Error list (all errors, warnings, and messages)

// 10. Identify all messages from CppCheck NOT identified in Visual Studio.

// For each message not in both:

// Identify the risk as: RISK or NOT RISK

// Identify which system (Visual Studio or CppCheck) found the issue

// Provide a couple of sentences describing the issue found

class C

{

std::set<int> typedefs;

bool is\_type(int type) const

{

if (typedefs.find(type) != typedefs.end())

return is\_type(type); // BUG: endless recursion

return false;

}

};

class A

{

int x;

A(const A& other) {}

};

class MySpecialType

{

public:

int MyVal = 1;

void DontThrow() noexcept

{

throw "Ha! I threw anyway!";

}

};

void foo(int\*\* a)

{

int b = 1;

\*a = &b;

}

void work\_with\_arrays(int count)

{

int buf[10];

if (count == 1000)

buf[count] = 0; // <- ERROR

}

void do\_something\_useless()

{

int sum = 0;

for (auto i = 0; i < 1000; ++i)

{

sum += i;

}

std::cout << "I summed up " << sum << " as the answer." << std::endl;

}

void vector\_test()

{

std::vector<int> items;

items.push\_back(1);

items.push\_back(2);

items.push\_back(3);

std::vector<int>::iterator iter;

for (iter = items.begin(); iter != items.end(); ++iter) {

if (\*iter == 2) {

items.erase(iter);

}

}

}

int a;

bool my\_function()

{

a = 1 + 2;

return a;

}

struct Token

{

Token\* next() { return nullptr; }

};

int foo(Token\* tok)

{

while (tok);

tok = tok->next();

return 0;

}

int main()

{

std::vector<int> counts{ 1, 2, 3, 5 };

int x = 0;

int y = 0;

int z = 0;

std::cout << "Welcome to the Questionable Code Test!" << std::endl;

//do\_something\_useless();

work\_with\_arrays(10);

assert(z = 2);

assert(my\_function() == 3);

try

{

int x = 5;

int y = 5;

int z = 5;

std::cout << "x + y + z = " << (x + y + z) << std::endl;

}

catch (...)

{

}

int\* c;

foo(&c);

vector\_test();

MySpecialType myobject;

std::cout << "myobject.MyVal = " << myobject.MyVal << std::endl;

}

// Run program: Ctrl + F5 or Debug > Start Without Debugging menu

// Debug program: F5 or Debug > Start Debugging menu