A Cppcheck Design

A.1 Introduction

The goal with this article is to give users an idea of how Cppcheck works.

Cppcheck is a static analysis tool that tries to completely avoid false warnings. A false warning is when the tool reports that there is an error even though there is no error.

Cppcheck is a relatively simple tool. I hope that this article will highlight that it is possible to avoid false warnings even with simple analysis.

A.2 Limitations of static analysis

There are many bugs in programs that are really hard to detect for tools. Here is an example:

```
    // calculate the number of days
    int days = hours / 23;
```

A human programmer knows that there are 24 hours in a day and therefore he could see that "23" is wrong. A tool will probably not know that there are 24 hours in a day.

A tool that tries to detect all bugs could write a warning message for every calculation in the program. Then it will correctly report that "hours / 23" is wrong but incorrectly warn about "hours / 24".

Cppcheck will only write a warning message if it can determine that the calculation is wrong. In this case, no error will be written.

A.3 Control flow analysis

When you review code you will probably use "control flow analysis" in your head to determine if there are bugs or not.

Control flow analysis is when you try to determine what the possible execution paths are.

The control flow analysis in Cppcheck is quite simple.

A.4 Buffer overflows

This is a simple description of how buffer overflows are detected by Cppcheck.

If an array is accessed out of bounds somewhere in its scope then an error message will be written. An example code:

```
void f()
{
    char a[10];
    if (x + y == 2) {
        a[20] = 0;
    }
}
```

Cppcheck will report this message:

```
Array 'a[10]' index 20 out of bounds
```

No control flow analysis is used. Cppcheck will not try to determine how execution can reach the "a[20] = 0;" statement. It is assumed that all statements are reachable. Cppcheck will detect the error even if it is really impossible that "x + y == 2" is true. I still claim that this is a correct warning because the statement is there and it has the error.

Cppcheck will also investigate function calls. But then control flow analysis can be needed to avoid false warnings. Here is an example that logically is the same as the previous example:

```
void f1(char *s)
{
    s[20] = 0;
}

void f2()
{
    char a[10];
    if (x + y == 2) {
        f1(a);
    }
}
```

Cppcheck will report this message:

```
Array 'a[10]' index 20 out of bounds
```

If the execution reaches the function call then there will be an error.

But if the condition is moved into "f1" then it will be necessary to prove that "x+y==2" can be true when the function is called from "f2". No error message is reported for this code:

```
void f1(char *s)
{
    if (x + y == 2) {
        s[20] = 0;
    }
}

void f2()
{
    char a[10];
    f1(a);
}
```

A.5 Memory leaks

The check uses simple control-flow analysis. The control flow analysis assumes that all conditions can always be either true or false. It is assumed that all statements are reachable. Here is an example:

```
void f()
{
    char *a = malloc(10);
    if (x + y == 2) {
        return;
    }
    free(a);
}
```

Cppcheck will determine that there is a leak at the "return;" statement:

```
Memory leak: a
```

Cppcheck doesn't try to determine how the execution reaches the "return;" statement. It will only see that if the execution reaches the "return;" then there will be a memory leak.

Lack of advanced control-flow analysis means that many bugs are not detected:

```
void f(int x)
{
    char *a = 0;

    if (x == 10)
        a = malloc(10);

    if (x == 20)
        free(a);
}
```

Cppcheck doesn't detect any error. The "all conditions can be either true/false" means that cppcheck doesn't know that "if (x==20)" is always false when "if (x==10)" is true. So Cppcheck can't establish that there is a leak. Many other static analysis tools will probably detect that there will be a leak if x is 10.

A.6 Final thoughts

You can not trust that Cppcheck will detect all bugs.

Cppcheck will just find some bugs. It is likely that you won't find these bugs unless you use Cppcheck.