Evaluation of Static Analysis Tool - CppCheck

Vimmi Taneja
Student of CS5393 (Fall 2016)
Texas State University
v_t28@txsstate.edu

ABSTRACT

C and C++ are unmanaged languages. The memory allocated by new or malloc must be freed manually by calling delete or free. If not freed, variables go out of scope and memory allocated to them is never freed and thus, memory is wasted and not available to other programs. This is unlike Java, C# which take care of garbage collection. Also, other issues which cause segmentation fault are arrays with index out of bounds and buffer overflow. Static analysis helps in finding such kind of bugs. In this paper, we talk about static analysis tool - Cppcheck for detecting memory leaks, buffer overflow, array index out of bounds etc. There are various tools available for static analysis – both commercial and free. Commercial include PC-Lint, Coverity etc. and free ones include cppcheck, CLang. One of the main reasons why Cppcheck tool was picked for this project is because it has very less false positives as compared to other free tools.

Keywords

Cppcheck, Static Analysis Tool, Software Quality

1. INTRODUCTION

Static analysis, also called static code analysis, is a method of computer program debugging that is done by examining the code without executing the program. The process provides an understanding of the code structure, and can help to ensure that the code adheres to industry standards.

Automated tools can assist programmers and developers in carrying out static analysis. It is usually fast (time of analysis close to time of compilation). There are various tools available for static analysis – some are free and some are commercial. Free tools include Cppcheck, FlawFinder, Rough Auditing Tool for Security (RATS), CppLint, CLang. Commercial tools include Parasoft Test and PVS-Studio. This project focuses on CppCheck tool because it is free, easy to install and claims zero false positives. The list of tools for static analysis can be found at https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#C.2C_C.2B.2B

In this paper, it will be explained how to install Cppcheck, overview of the tool, the programs that we used to run it and strengths and weaknesses of the tool.

2. MOTIVATION

Memory leaks and segmentation faults are very common and show blocking errors in C++. Since C++ is unmanaged language, these errors are more encountered in it. These errors are difficult to detect and cause the program to terminate. Other issues are buffer overflow, boundary conditions, etc.

Following are the various important issues and vulnerabilities in software.

2.1 Memory leak

A memory leak occurs when a piece (or pieces) of memory that was previously allocated by a programmer is not properly de-allocated by the programmer. Even though that memory is no longer in use by the program, it is still "reserved", and that piece of memory cannot be used by the program until it is properly de-allocated by the programmer. That's why it's called a memory leak – because it's like a leaky faucet in which water is being wasted, only in this case its computer memory.

The problem caused by a memory leak is that it leaves chunk(s) of memory unavailable for use by the programmer. If a program has a lot of memory that hasn't been de-allocated, then that could slow down the performance of the program. If there's no memory left in the program because of memory leaks, then that could of course cause the program to crash.

2.2 Buffer overflow

The buffer overflow is the vulnerability which arises when a program attempt to put more data in a buffer than it can hold. It is considered as a severe vulnerability because this vulnerability lead to the program crashes and it may put the program into infinite loops. Sometimes, it can also be used by an attacker to execute the arbitrary code, which is outside of program scope. The mostly effected languages with this type of problem are C and C++. Other languages, like Java, C# are more secure because they have bounds checking, have native string type and most importantly they prohibit direct memory access. Most of the overflow (up to 70%) occurs due to improper use of C library functions, so it's best to trace the function calls (strcpy, strcat, sprint etc.) at its entry time. Besides, using unsafe function, the reasons for buffer overflow could be by looping over an array using an index that may be too high, or maybe through integer errors. The buffer overflows can be stack overflow or heap overflows. Examples of some damages done by buffer overflows are – Code red worm: estimated loss worldwide 2.62 billion USD, Sasser worm: this shut down x-ray machines at a Swedish hospital and caused delta airlines to cancel several transatlantic flights. Similarly, the Morris worm (1988) and Slammer (2003) also had a high impact in the past. In 2004, about 20% published exploits reported by US-CERT involved buffer overflow.

2.3 Format String

Format String handling is a quite common problem in C programs. Mitre ranked it in 9th position in their dangerous list of vulnerability in between the years 2001 and 2006. Let's look the example to understand the problem. E.g. printf (“My number is: %d”, 786); the output of this statement looks like: My number is: 786. Here, %d acts as a format string and the function behavior is controlled by it. The function retrieves the parameters requested by the format string from the stack. So, the problem arises if there is a mismatch between the format string and the actual arguments. For instance printf (“My number is %d, your number is %d”, mynumber); here actually program asks for 2 arguments but we provide only one i.e. mynumber. The printf () function usually takes variable length of arguments.

So, the program looks fine. To find the mismatch, compilers need to know how printf works and what its meaning is. Since format string needs 2 arguments, it will fetch 2 data items from the stack. Unless it is a mark with a boundary, printf will continue fetching data from the stack. Thus, trouble get started when printf () starts fetching data in uncontrolled way. It’s because this could may allow attacker to view the arbitrary stack content (e.g. printf (“%08x %08x %08x”)), crash the program (e.g. printf (“%s%s%s%s%s%s”)), view the memory at any location (e.g. printf (%s)) etc.

The static analysis process helps in finding out such issues in early stage. Also, it helps in early detection of vulnerabilities in the code base and hence resolve security issues by tracing the taint data from source to sink or analyzing patterns. This could provide ease to fix problems that can be very vital and costly later.

3. TOOL OVERVIEW AND ANALYZED PROBLEMS

3.1 Overview of the tool

Cppcheck is a static analysis tool for C and C++ code that works by matching source file tokens against a pattern for each analysis rule. It is open-source, free, cross-platform, and easy-to-use. Unlike C/C++ compilers and many other analysis tools it does not detect syntax errors in the code. It primarily detects the types of bugs that the compilers normally do not detect. The goal is to detect only real errors in the code (i.e. have zero false positives).

This tool is distributed under the GNU General Public License. Created by Daniel Marjamäki, CPPCheck offers a command line mode as well as a GUI mode and has several possibilities for environment integration.

Daniel Marjamäki is the project's manager. The project's source code can be downloaded from the github website.[2]

3.1.1 Clients and Plugins

The version of CppCheck used is 1.76.1. It is integrated with many popular development tools. For instance: Eclipse, Hudson, Jenkins, Mercurial, Tortoise SVN, Visual studio, Code::Blocks, CppDepend5.

3.1.2 Features

Detect various kinds of bugs in your code.

- Out of bounds checking
- Memory leaks checking
- Detect possible null pointer dereferences
- Check for uninitialized variables
- Check for invalid usage of STL
- Checking exception safety
- Warn if obsolete or unsafe functions are used
- Warn about unused or redundant code
- Detect various suspicious code indicating bugs

For a list of all checks see: http://sourceforge.net/p/cppcheck/wiki/ListOfChecks.

Both command line interface and graphical user interface are available.

3.1.3 Scope of Use

One of the basic advantages of the Cppcheck analyzer is that it is easy-to-use. It is good for teaching, and studying, the static analysis methodology: for instance, you install Cppcheck on a Windows system and get a GUI interface allowing you to immediately start checking your projects.

3.1.4 User Interface

User interface of Cppcheck on windows is shown in the picture. There is a “Check” link in menu. Click on “Check” and a path to directory or a file can be specified that needs to be checked.

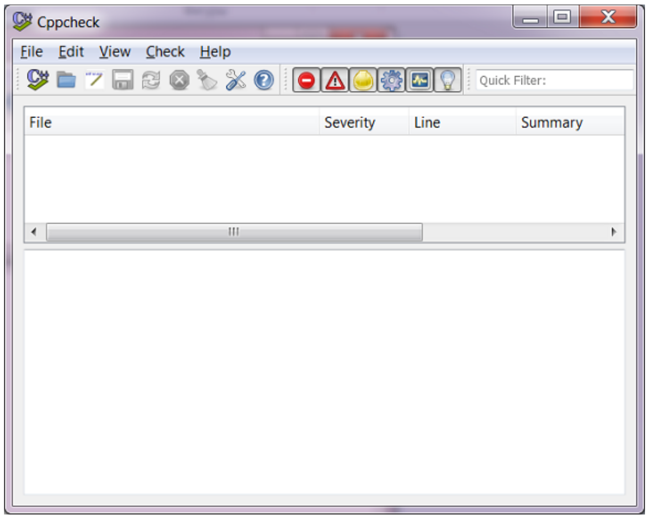
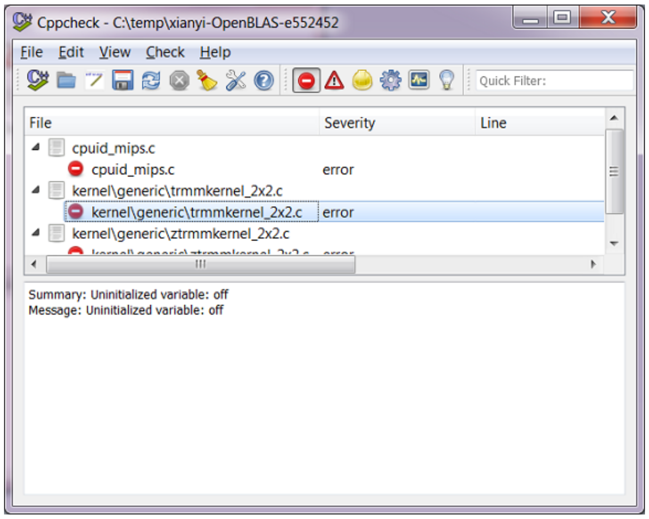


Figure 1. Cppcheck for Windows, the main window

The project analysis report looks something like the screenshot in Figure 2.

When analysis is over, you can study the diagnostic messages. They are grouped into the following categories: Errors, Warnings, Style Warnings, Portability Warnings, Performance Warnings, Information Messages. You can easily turn these groups on and off, by clicking on special buttons on the toolbar.

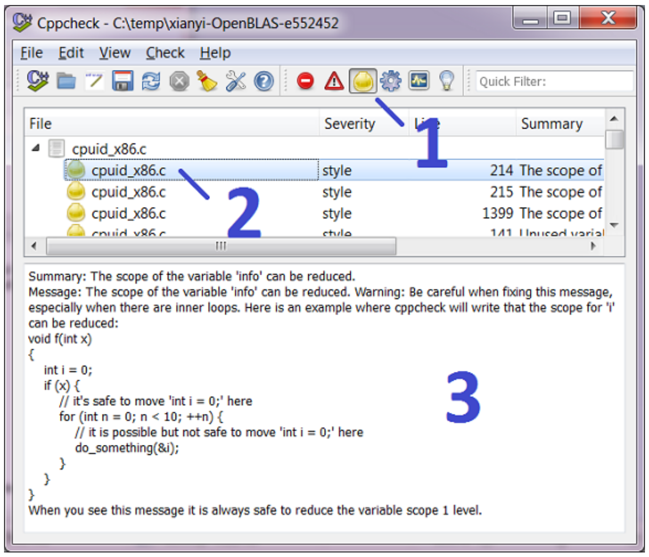

Cppcheck - C:\temp\xianyi-OpenBLAS-e552452
File Edit View Check Help
Quick Filter:

File	Severity	Line
cpuid_mips.c		
cpuid_mips.c	error	
kernel\generic\trmmkernel_2x2.c		
kernel\generic\trmmkernel_2x2.c	error	
kernel\generic\ztrmmkernel_2x2.c		

```
Summary: Uninitialized variable: off
Message: Uninitialized variable: off
```

Figure 2. Project Analysis Report

Figure 3 shows how to set the message view mode to see only Style Warnings: the message group "Style Warnings" is on, while all the rest are off (1). The file "cpuid_x86.c" contains several warnings of this type, and the first one is selected which refers to line 214 (2). The diagnostic's description is shown in the lower window (3).


Cppcheck - C:\temp\xianyi-OpenBLAS-e552452
File Edit View Check Help
Quick Filter:
1
2
3

File	Severity	Line	Summary
cpuid_x86.c			
cpuid_x86.c	style	214	The scope of
cpuid_x86.c	style	215	The scope of
cpuid_x86.c	style	1399	The scope of
cpuid_x86.c	style	141	Unused varial

```
Summary: The scope of the variable 'info' can be reduced.
Message: The scope of the variable 'info' can be reduced. Warning: Be careful when fixing this message,
especially when there are inner loops. Here is an example where cppcheck will write that the scope for 'i'
can be reduced:
void f(int x)
{
    int i = 0;
    if (x) {
        // it's safe to move 'int i = 0;' here
        for (int n = 0; n < 10; ++n) {
            // it is possible but not safe to move 'int i = 0;' here
            do_something(&i);
        }
    }
}
When you see this message it is always safe to reduce the variable scope 1 level.
```

Figure 3. Setting up the message view mode

Cppcheck needs some customization. If you start using it on a deeper level, you'll need to customize some settings. For example, you'll need to specify paths to third-party libraries, integrate Cppcheck with your development environment, or set up night checks. But the fact that you can just select a directory and get a result, is just awesome! It's especially so, if you you're only getting started with static analysis, in which case such a capability is invaluable.

To view the problematic statement, just double click on diagnostic message, the tool will open that file in notepad and point to that line. Also, if there is an issue regarding tool, it can be reported on their website.

3.2 Programs and inputs

We wrote our own code to test tool's capabilities and got some open source programs from github. We checked the following 5 open source projects:

Eigen: Eigen is a high-level C++ library of template headers for linear algebra, matrix, and vector operations, numerical solvers, and related algorithms. Number of files = 737

OpenDDS. The Data Distribution Service for Real-Time Systems (DDS) is an Object Management Group (OMG), machine to machine middleware "m2m" standard, which aims to enable scalable, real-time, dependable, high performance and interoperable data exchanges between publishers and subscribers.

Wild Magic 5. C++ library for real-time computer graphics and physics, mathematics, geometry, numerical analysis, and image analysis.

Mplayer - MPlayer is a movie player which runs on many systems.

Synergy - Synergy is a software application for sharing a keyboard and mouse between multiple computers.

All the above projects had number of files between 260 and 930.

Our own code consists of small functions with faults or errors hand seeded [7]. All the above programs are input to the tool and output is set of diagnostic messages.

4. STUDY

4.1 CppCheck - Tool Installation

Installation is straightforward. Cppcheck comes as drop-in binary from http://cppcheck.sourceforge.net/ for both 32 bit and 64 bit CPUs. Run the executable. It will install all the files in C:\ProgramFiles\CppCheck folder. It contains cppcheck.exe and cppCheckGUI.exe. Click on GUI executable to run GUI version.

To integrate into Code::Blocks, install cppcheck from its binary, then go to Settings -> Environment and provide path of cppcheck exe in configuration of cppcheck.

CppCheck does not fully parse the code, but scans the actual text of the source files. Thus, it can be run on any file.

Supported code and platforms:

- You can check non-standard code that includes various compiler extensions, inline assembly code, etc.
- Cppcheck should be compilable by any C++ compiler that handles the latest C++ standard.

The possible severities for messages are:

- Error - used when bugs are found
- Warning - suggestions about defensive programming to prevent bugs
- Style - stylistic issues related to code cleanup (unused functions, redundant code, constness, and such)
- Performance- Suggestions for making the code faster. These suggestions are only based on common knowledge. It is not certain you'll get any

measurable difference in speed by fixing these messages.

- Portability - portability warnings. 64-bit portability. code might work different on different compilers. etc.
- Information - Configuration problems. The recommendation is to only enable these during configuration.

4.2 Analyzing the problems using the tool

We tested for Memory leak, array index out of bounds, buffer access out of bounds, unreachable code, unused functions, unread variables, unused Allocated memory, Negative array index, negativeMemoryAllocationSize by malloc function, null pointer dereference, division by zero, deallocating a deallocated pointer, Constructors which should be explicit, mismatch of allocation and deallocation, increment boolean, compare bool expression with integer other than 0 or 1, missing constructors by writing our own faulty code.

Constructors should initialize member variables. If the constructor is missing, member variables of builtin types are left uninitialized when the class is instantiated. That may cause bugs or undefined behavior.

We tested 17 of the overall total claims done by the tool. They fall into following categories:

Boolean, Bounds checking, Class, exception safety, Memory leaks, null pointer dereferencing, STL usage, string, unitialized variables, unused functions, unused variables.

We believe that these represent a good subset of the overall number and kinds of errors detected by the tool and the claims made by the developers of Cppcheck.

4.2.1 *Sample programs*

```
void foo(char* str)
{
   char* buf = new char[8];
   strcpy(buf, str);

   FILE* file = fopen("out.txt", "w");
   if(!file)
      return;
        for(char* c = buf; *c; ++c)
                fputc((int)*c. file);
        delete buf;
}
```

Cppcheck detected 3 errors in above program –

Memory leak at line 8 where program returns without releasing memory for buf object

Mismatching alloacation and deallocation at line 13, it should be delete[] buf

Resource leak – at last line, when the program terminates without closing the file. File is resource which is not being released.

```
void foo(char* str)
 {
    char buf[8];
    strcpy(buf, "Hello world");
    printf("foo says %d\n", buf);
 }
```

The tool detected 1 error and gave 1 warning in above program.

Error – Buffer is accessed out of bounds since buffer is 8 characters and string is more than 8 characters.

Warning - %d in format string (no. 1) requires 'int' but the argument type is 'char *' in printf.

All the above errors and warnings are true positives.

```
void memLeak( )
{
    // OK
    int * p = new int;   // unused allocated
memory
    delete p;

    // Memory leak
    int * q = new int;
    *q = 15;
      // no delete .. memory leak
```

```
    delete p;  // double free

    int *sortArray;
    sortArray = new int[5];
    // mismatch allocation and deallocation
    delete sortArray; // should be delete[]

}
```

Tool detected 4 errors and 2 warnings.

Errors - Memory pointed to by 'p' is freed twice when we delete p again at line 12. Memory leak for ‘q’ because there is no delete for q. Deallocating a deallocated pointer: p, Mismatching allocation and deallocation: sortArray. Deallocation of sortArray must have been done by delete[] not by delete.

Warnings - Variable 'p' is allocated memory that is never used. Variable 'sortArray' is allocated memory that is never used.

```
    char a[2];
    a[2] = 'c';
```

In above code, array index is out of bounds. Array size is 2 characters and we are assigning third element a value. Tool detects it.

```
int testDivide(int a , int b)
{
    return a/b;
}
// divide by zero
     testDivide(3, 0);
```

In above code, we are deliberately dividing by zero to test the tool's capability. It successfully detected it.

Following is the source code from open source project *Eigen*

```
FILE* file = fopen(filename, "r");
  if (!file) {
    return false;
  }
  if    (1    !=    fread(&max_clock_speed,
sizeof(max_clock_speed), 1, file)) {
    return false;
  }
```

Tool detected resource leak in above code since file is not closed and file handle is not released even though variable file has gone out of scope.

OpenDDS

Errors – Null pointer dereference. Following is the code snippet from NetworkAddress.cpp in OpenDDS.

```
ACE_INET_Addr *if_addrs = 0;
..
#ifdef OPENDDS_SAFETY_PROFILE
    -1;
#else
    ACE::get_ip_interfaces(if_cnt,
if_addrs);
#endif
…
 if (if_addrs[j].is_loopback())
```

The last line where if_addrs is referenced, is null pointer dereference per cppcheck.

Memory leak: Following is the code snippet from AbstractionLayer.cpp in OpenDDS. It allocates memory by calling new but does not call delete, so it is memory leak.

```
DistributedContent::FileDiffTypeSupportImpl*
fileinfo_dt =
                new
DistributedContent::FileDiffTypeSupportImpl(
);
```

Warning – Member variable not initialized in the constructor.

Style - The scope of the variable can be reduced. Class has constructor with 1 argument that is not explicit.

Performance – Prefer prefix operators for non-primitive types. Function parameter which is passed by value should be passed by const reference which is quicker in C++.

4.3 Results and Analysis

We tested with our own faulty code, found that most of the errors or warnings mentioned were being detected. Some of the checks were not clear and we were not able to verify. For example,

strncpy() leaving string unterminated – If the string was unterminated, cppcheck did not give any error.

Following is the summary of the results of running cppcheck on open source projects. Tool was quick in scanning the files. We verified some of the errors, they seemed like true positives.

Table 1: Open source projects' results

Project	Files checked	Errors	Warnings	Duration(s)
Synergy	262	3	13	13
Wild magic 5	782	3	232	27
Eigen	737	12	65	31
OpenDDS	931	4	20	52
Mplayer	648	191	559	212

The above table shows number of files in each project, number of errors and warning messages displayed by the tool and time taken by tool in evaluating the project. We did not consider other kinds of messages such as information, portability and style as those are not bugs.

Types of errors in above projects –

1. Synergy - memory freed twice, Null pointer dereference, Syntax error
2. Wild Magic 5 - Array index out of bounds
3. Eigen - syntax error, resource leak.
4. OpenDDS - Null pointer dereference, Memory leak
5. MPlayer - Common realloc mistake: 'buf' nulled but not freed upon failure, shifting a negative value is undefined behavior, Division by zero, Resource leak, Memory leak

We looked at the code and the lines of error, they were true positives. The results tell that cppcheck is good at finding bugs and quick enough to check 700 files.

4.3.1 Strengths

Advantages: The tool is free, easy to install, available as plugin and can be integrated into other editors. Its easy to use, just give the file path or directory path and run and quickly we can see list of errors, warnings, style issues, information, portability issues. Also, double click on diagnostic message and it opens the code in notepad pointing to the line of error. It has very low false positives and checks lot of issues [1]. It tells you not only about incorrect code (like memory leaks or using uninitialized variables) but also about style issues (e.g. it will suggest making a C++ class constructor explicit if it contains one argument etc.). A unique property of cppcheck is that it can be run on stand-alone files. The advantage is that it's easy to use (you can, for example, check unix or mac source code on windows (and vice-versa)). GUI is also easy to use. There is a way to just choose messages like only errors to be displayed. Error messages are clear and easy to understand as compared to Visual studio error messages which are vague and not comprehensible.

- Plugins and integrations for several IDE: Eclipse, Hudson, Jenkins, Visual Studio.
- Daniel's plan is to release a new version every other month or so, and he's been keeping up with that goal.
- Easily applicable to any file and ability to scan all files even outside the build.
- Available in many world languages, including English, Dutch, Finnish, Swedish, German, Russian, Serbian and Japanese.

4.3.2 Weaknesses

- Doesn't detect many bugs (as with most of the other tools) since it has less information to use.
- Customization requires good deal of effort
- Issues detected are shallower than those detected by other tools.

4.4 Suggested Improvements

Tool can be enhanced to detect more errors as other tools such as PVS-Studio can detect.

5. CONCLUSION

As a summary, we believe that Cppcheck is a useful first line of defense with very low run time overhead. Its key role is quick scanning of code under development. Its free and easy to use and detects important issues in unmanaged languages like C++. Hence, it should be used by every programmer to maintain quality of code.

6. REFERENCES

[1] https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/

[2] The project's website: http://cppcheck.sourceforge.net/

[3] MPlayer - http://www.mplayerhq.hu/design7/info.html

[4] Eigen - http://eigen.tuxfamily.org/index.php?title=Main_Page

[5] OpenDDS- https://github.com/objectcomputing/OpenDDS/releases

[6] WildMagic 5 - https://github.com/bazhenovc/WildMagic

[7] Custom source code: at Athena.cs.txstate.edu/home/Students/v_t28/CS 5393