An Overview of Static Analyzers for C/C++ Code

Aleksandr Alekseev (/en/b/a/aleksandr-alekseev)
Articles: 1

11.05.2016

Contents

m/Code_Analysis)

ɔurner.com/viva64-

C and C++ programmers tend to make mistakes when writing code.

Many of these mistakes can be found using *-Wall*, asserts, tests, meticulous code review, IDE warnings, building with different compilers for different operating systems running on different hardware configurations, and the like. But even all these means combined often fail to reveal all the bugs. Static code analysis helps improve the situation a little. In this post, we will take a look at some static analysis tools. **[The author of this article is not an employee of our company, and his opinion may be different from ours.]**

Cppcheck

Cppcheck is a free open-source (https://github.com/danmar/cppcheck/) (GPLv3) cross-platform static analyzer, which comes as an out-of-the-box solution with many *nix systems. It can also integrate with many IDEs. At the time of writing this article, Cppcheck is a living, evolving project.

Example of use:

```
cppcheck ./src/
```

Example of output:

```
[some.c:57]: (error) Common realloc mistake: 'numarr' nulled
but not freed upon failure

[some.c:154]: (error) Dangerous usage of 'n'
(strncpy doesn't always null-terminate it)
```

A good thing about Cppcheck is that it is pretty fast. There are no reasons why you shouldn't add it into your continuous integration system (http://eax.me/jenkins/) so that you could fix each and every warning it generates - even despite that many of these turn out to be false positives.

Official website: http://cppcheck.sourceforge.net/ (http://cppcheck.sourceforge.net/)

Clang Static Analyzer

Another free open-source cross-platform static analyzer, which comes as a part of so called "LLVM-stack". Unlike Cppcheck, Clang Static Analyzer is much slower, but it can catch much more critical bugs.

Example of forming an analysis report for PostgreSQL (http://eax.me/postgresql-build/) project:

```
CC=/usr/local/bin/clang38 CFLAGS="-O0 -g" \
  ./configure --enable-cassert --enable-debug
gmake clean
mkdir ../report-201604/
/usr/local/bin/scan-build38 -o ../report-201604/ gmake -j2
```

Example of forming an analysis report for FreeBSD kernel (http://eax.me/freebsd-kernel-world/):

```
# using custom MAKEOBJDIR allows building the kernel
# under a non-root account
mkdir /tmp/freebsd-obj
# the build:
COMPILER_TYPE=clang /usr/local/bin/scan-build38 -o ../report-201604/ \
  make buildkernel KERNCONF=GENERIC MAKEOBJDIRPREFIX=/tmp/freebsd-obj
```

The idea behind it, as you can easily guess, is to clean up the project and then start the build under scan-build.

What you get as the output is a neat HTML report with highly detailed comments, bug filtering by type, and so on. I do recommend that you visit their site for some examples of how it is done.

Since we started talking about it, I can't but mention that the Clang/LLVM infrastructure also provides *dynamic* analysis tools, known as "sanitizers". There are lots of them; they can catch very tough bugs and run faster than Valgrind (http://valgrind.org/docs/manual/quick-start.html) (although exclusively on Linux). Unfortunately, discussing sanitizers is beyond the scope of this article, so I recommend that you read about these tools on your own (http://clang.llvm.org/docs/MemorySanitizer.html).

Official website: http://clang-analyzer.llvm.org/ (http://clang-analyzer.llvm.org/)

PVS-Studio

A proprietary, commercially distributed static analyzer. PVS-Studio runs only on Windows and only with Visual Studio. There is much evidence that a Linux version exists, but you won't find such a version at the official website. As far as I understand, the license price is discussed individually with every customer. A trial version is also available.

I was testing PVS-Studio 6.02 on Windows 7 SP1 running under KVM (http://eax.me/kvm/) with Visual Studio 2013 Express Edition. During the PVS-Studio installation, .NET Framework 4.6 was additionally downloaded. This is how the analysis is done: you open a project (I picked PostgreSQL) in Visual Studio, then click on "I'm about to start the build" in PVS-Studio, go to Visual Studio and click on "Build", and when the build is done, you go back to PVS-Studio and click on "Finished" and open the report.

PVS-Studio does know how to catch very tough bugs, which Clang Static Analyzer can't (example (http://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=58666ed28ab59a2686ee08bc648b4e9959aacfce)). I was also impressed by the interface, which allows you to sort and filter bugs by type, severity, location file, and so on.

On the one hand, it's a pity that your project must be compilable under Windows for you to be able to scan it with PVS-Studio. On the other hand, using CMake with your project to build/test it on various operating systems, including Windows, is a nice idea anyway. So, I guess, the tool being designed for Windows isn't that much of a disadvantage. Besides, here are a few links to some tips from people who managed to run PVS-Studio over projects that don't compile under Windows: one (/en/b/0377/), two (/en/b/0299/), three (/en/b/0387/), four (https://github.com/aaptel/pvs-tool).

Official website: http://www.viva64.com/en/pvs-studio/ (/en/pvs-studio/)

Coverity Scan

Coverity is believed to be one of the most sophisticated (and, therefore, expensive) static analyzers. Unfortunately, there is not even a trial version available at the official website. You can try filling out a special form, and in case you work for IBM, you might be lucky enough to hear from them. If you are *very* eager to get Coverity, you may find some prehistoric versions through unofficial channels. Coverity is available both for Windows and Linux and relies on a similar principle as PVS-Studio. Unlike the latter, though, Coverity will never let you view the report without a key or crack. And to find one or the other, you must try not just very hard, but *extremely* hard.

Fortunately, Coverity is also available as a SaaS version, which is known as Coverity Scan. Not only is it available to mere mortals, but it is also absolutely free. Coverity Scan is not bound to any particular platform, but you are allowed to use it only on open-source projects.

This is how it works. You register your project through the web interface (or join an already existing project, but this case is not that interesting). For you to be allowed to view the analysis results, your project needs to be approved by an administrator; this procedure takes 1-2 business days.

Analysis reports are formed in the following way. First you build your project locally using special utility Coverity Build Tool, which is similar to scan-build by Clang Static Analyzer and runs on all existing platforms, including various exotic ones such as FreeBSD and even NetBSD.

Coverity Build Tool installation:

```
tar -xvzf cov-analysis-linux64-7.7.0.4.tar.gz
export PATH=/home/eax/temp/cov-analysis-linux64-7.7.0.4/bin:$PATH
```

Preparing a test project (I was using the code discussed in the post Going on with Exploration of OpenGL: Simple Text Output (RU) (http://eax.me/opengl-text/):

```
git clone git@github.com:afiskon/c-opengl-text.git
cd c-opengl-text
git submodule init
git submodule update
mkdir build
cd build
cmake ..
```

Then we build the project with cov-build:

```
cov-build --dir cov-int make -j2 demo emdconv
```

Important! Do not change the name of the cov-int directory.

Pack the cov-int directory into an archive:

```
tar -cvzf c-opengl-text.tgz cov-int
```

Then you upload the archive through the Upload form as a Project Build. See instructions at the Coverity Scan site on automating this step using curl. Wait a little, and you can finally view the analysis results. Note that you have to send at least one build for analysis to have it approved by administrators.

Coverity Scan is very good at catching bugs - surely better than Clang Static Analyzer. It produces false positives as well, but there are much fewer of them. The web interface provides a convenient feature, kind of an integrated bug tracker, which allows you to assign different severity levels to bugs, or developers to address them, and so on. It also shows which errors are new and which were already present in the previous builds. Finally, you can mark false positives and hide them.

Note that you don't have to be the owner of a project to get it analyzed by Coverity Scan. For example, I managed to successfully analyze the code of PostgreSQL without joining the already existing project. I suspect that if you really want and try hard enough, you could pass off a bit of not quite open-source code for analysis (for example using Git submodules (http://eax.me/git-commands/)).

Official website: https://scan.coverity.com/ (https://scan.coverity.com/)

Conclusion

Here are two more static analyzers, which I didn't elaborate on here:

- http://www.splint.org/ (http://www.splint.org/);
- http://oclint.org/ (http://oclint.org/);

Each of the analyzers discussed above can catch bugs that none of the others can; so, ideally, you want to use them all at once. In reality, it's hardly possible to maintain this practice all the time for objective reasons; yet giving them at least one run on your project before a release is certainly a good idea. Clang Static Analyzer, however, seems to be the most universal and rather powerful at the same time. If you are looking for one analyzer to use with every project, pick that one. However, I'd still recommend using at least PVS-Studio or Coverity Scan in addition.

What static analyzers have you tried and/or used regularly and what are your impressions of them?

P.S.

This article was written by Russian blogger Aleksandr Alekseev, who kindly permitted us to post it together with the translated version at our site. The original article can be found here: http://eax.me/c-static-analysis/ (http://eax.me/c-static-analysis/)

Use PVS-Studio to search for bugs in C, C++, and C# code

We offer you to check your project code with PVS-Studio. Just one bug found in the project will show you the benefits of the static code analysis methodology better than a dozen of the articles.

goto PVS-Studio; (/en/pvs-studio/?utm_source=viva&utm_medium=goto_bottom&utm_campaign=home

Previous article (/en/b/0396/) Next article (/en/b/0398/)

Aleksandr Alekseev (/en/b/a/aleksandr-alekseev)
Articles: 1

Liked this article? Read these! (/en/b/?utm_source=viva&utm_medium=recent_posts_bottom&utm_campaign=blog)

(/en/b/0563/)

Why embedded developers should use static code analysis

(/en/b/0562/)

One Doesn't Simply Edit Subtitles

How many people use subtitles worldwide? Probably, a lot. In the

(/en/b/0561/)

Static Code Analyzer PVS-Studio 6.22 Now Supports ARM Compilers (Keil, IAR)

I decided to briefly highlight 3 reasons why static analysis tools for program code may be useful for embedded developers.

(/en/b/0563/)

Internet you can find subtitles for almost any film ...

(/en/b/0562/)

PVS-Studio is a static code analyzer detecting errors and potential vulnerabilities in the code of applications written in C, C++, ...

(/en/b/0561/)

All articles → (/en/b/?utm_source=viva&utm_medium=all_articles_bottom&utm_campaign=blog)

PVS-Studio

We develop a PVS-Studio static code analyzer that finds errors in the C, C++, and C# programs on Windows and Linux.

goto PVS-Studio; (/en/pvs-studio/)

PVS-Studio

Buy

Achievements

Interesting

Company

Contact Us (/en/about-feedback/)

(https://twitter.com/Code_Analysis)

(http://feeds.feedburner.com/viva64-blog-en)

Site search

Rus (/ru/b/0397/)

Orphus Ctrl+Enter (http://orphus.ru)