meta-codechecker

Static analysis for your project and how it works

Jan-Simon Möller, dl9pf@gmx.de
Intro

Dipl.-Ing. Jan-Simon Möller
dl9pf on freenode
dl9pf@gmx.de

AGL Release Manager
jsmoeller@linuxfoundation.org

OpenEmbedded & Yocto Project Board Member
Topics

- Static Analysis - whaaaaat?
- Overview of tools
- CodeChecker
- meta-codechecker
- Summary
- Q/A
Static Analysis - whaaaaat?

& why you should use it!
Static Analysis - whaaaat?

- Static Analysis is a method to analyse a program that is performed without actually executing programs.
- Static Analysis becomes an increasingly important topic when the project involves Functional Safety aspects. This is the case in Automotive and in Automation as well.

"But of course /MY code is always correct. "
- But the auditor needs a way to (ap)prove that!
Motivation

- Static analysis will not solve all problems (™).
- It will help catching some (possibly tricky to find) bugs.
- The goal is to show ways how to do this using open source tools available.
- I will introduce basics but focus on what can be integrated with OpenEmbedded / The Yocto Project builds.
Overview of static analysis tools
Overview of tools

- There are tools available as OSS and proprietary tools.
- Some do pattern recognition, some use/enhance compilers, some are simple scripts. OSS tools include:
  - gcc
  - clang
  - cppcheck
  - flawfinder
  - rats
  - split
During development you can easily use these directly within your source tree:

- **gcc (since gcc 10)**
  - gcc -fanalyzer

- **clang**
  - e.g. scan-build make

- **cppcheck**

  *gcc -fanalyzer enables:*
  - `-Wanalyzer-double-fclose`
  - `-Wanalyzer-double-free`
  - `-Wanalyzer-exposure-through-output-file`
  - `-Wanalyzer-file-leak`
  - `-Wanalyzer-free-of-non-heap`
  - `-Wanalyzer-malloc-leak`
  - `-Wanalyzer-possible-null-argument`
  - `-Wanalyzer-possible-null-dereference`
  - `-Wanalyzer-null-argument`
  - `-Wanalyzer-null-dereference`
  - `-Wanalyzer-stale-setjmp-buffer`
  - `-Wanalyzer-tainted-array-index`
  - `-Wanalyzer-unsafe-call-within-signal-handler`
  - `-Wanalyzer-use-after-free`
  - `-Wanalyzer-use-of-pointer-in-stale-stack-frame`
gcc

> gcc -Werror -fanalyzer nullpointer.c

nullpointer.c: In function 'main':
nullpointer.c:7:5: error: dereference of NULL 'pointer' [CWE-690] [-Werror=analyzer-null-dereference]
  7 | int value = *pointer; /* Dereferencing happens here */
       ^~~~~

'main': events 1-2

| 6 | int * pointer = NULL;
    ^~~~~~~
(1) 'pointer' is NULL

| 7 | int value = *pointer; /* Dereferencing happens here */
   ^~~~~
(2) dereference of NULL 'pointer'

ccl: all warnings being treated as errors
clang (clang-tidy)

> clang-tidy nullpointer.c  
Running without flags.  
2 warnings generated.

nullpointer.c:7:5: warning: Value stored to 'value' during its initialization is never read 
[clang-analyzer-deadcode.DeadStores]
int value = *pointer; /* Dereferencing happens here */
  ^
nullpointer.c:7:5: note: Value stored to 'value' during its initialization is never read

nullpointer.c:7:13: warning: Dereference of null pointer (loaded from variable 'pointer')  
[clang-analyzer-core.NullDereference]
int value = *pointer; /* Dereferencing happens here */
  ^

nullpointer.c:6:1: note: 'pointer' initialized to a null pointer value
int * pointer = NULL;
  ^

nullpointer.c:7:13: note: Dereference of null pointer (loaded from variable 'pointer')
int value = *pointer; /* Dereferencing happens here */
  ^
clang (scan-build)

> scan-build make

scan-build: Using '/usr/bin/clang-10.0.1' for static analysis
/usr/bin/ccc-analyzer  -c nullpointer.c -o nullpointer

nullpointer.c:7:5: warning: Value stored to 'value' during its initialization is never read
  int value = *pointer; /* Dereferencing happens here */
  ^~~~~

nullpointer.c:7:13: warning: Dereference of null pointer (loaded from variable 'pointer')
  int value = *pointer; /* Dereferencing happens here */
  ^~~~~~~

2 warnings generated.
scan-build: 2 bugs found.

> scan-view /tmp/scan-build-2020-10-15-161857-10509-1
Starting scan-view at: http://127.0.0.1:8181

(-- point browser to this)
cppcheck

> cppcheck nullpointer.c
Checking nullpointer.c ...
nullpointer.c:7:14: error: Null pointer dereference: pointer [nullPointer]
  int value = *pointer; /* Dereferencing happens here */
    ^
nullpointer.c:6:17: note: Assignment 'pointer=NULL', assigned value is 0
  int * pointer = NULL;
    ^
nullpointer.c:7:14: note: Null pointer dereference
  int value = *pointer; /* Dereferencing happens here */
    ^
CodeChecker

https://github.com/Ericsson/codechecker

Collection of tools to

- intercept and log the build calls
- analyse the gathered data using (clang-tidy and clangSA)
- report (static or webui)

Extension and successor of the original clang static analyser / scan-build.
<table>
<thead>
<tr>
<th>Name</th>
<th>Number of unresolved reports</th>
<th>Detection status</th>
<th>Analyzer statistics</th>
<th>Storage date</th>
<th>Analysis duration</th>
<th>Check command</th>
<th>Version tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agl-service-gps@oneshot</td>
<td>1</td>
<td>🟥 (1)</td>
<td>clangsa: ✗ (1)</td>
<td>2020-07-02</td>
<td>08:41:01</td>
<td>Show</td>
<td>6.13</td>
<td>(db15618c00 b2641197d8 fa2f1599a37 58909924)</td>
</tr>
<tr>
<td>cynagora@oneshot</td>
<td>17</td>
<td>🟥 (17)</td>
<td>clang-tidy: ✗ (30)</td>
<td>2020-07-02</td>
<td>08:00:16</td>
<td>Show</td>
<td>6.13</td>
<td>(db15618c00 b2641197d8 fa2f1599a37 58909924)</td>
</tr>
<tr>
<td>app-framework-binder@oneshot</td>
<td>79</td>
<td>🟥 (79)</td>
<td>clangsa: ✗ (92)</td>
<td>2020-07-02</td>
<td>07:50:44</td>
<td>Show</td>
<td>6.13</td>
<td>(db15618c00 b2641197d8 fa2f1599a37 58909924)</td>
</tr>
<tr>
<td>app-framework-main@oneshot</td>
<td>35</td>
<td>🟥 (36)</td>
<td>clang-tidy: ✗ (34)</td>
<td>2020-07-01</td>
<td>22:04:52</td>
<td>Show</td>
<td>6.13</td>
<td>(db15618c00 b2641197d8 fa2f1599a37 58909924)</td>
</tr>
<tr>
<td>agl-service-audiomixer</td>
<td>4</td>
<td>🟥 (4)</td>
<td>clang-tidy: ✗ (2)</td>
<td>2020-07-01</td>
<td>21:36:00</td>
<td>Show</td>
<td>6.13</td>
<td>(db15618c00 b2641197d8 fa2f1599a37 58909924)</td>
</tr>
</tbody>
</table>
CodeChecker usage

- Userspace tool CodeChecker is a set of python helpers
  - main feature is that you wrap your build commands like so
    - CodeChecker log -b "make" -o compilation.json
  - This will preload a logger and store the compiler commands
  - With the exact commands logged, we can replay the compilation using clang and its tools clang-tidy and clangSA
    - CodeChecker analyze compilation.json -o ./reports
CodeChecker usage #2

● From there you can 'parse' into reports
  ○ CodeChecker parse ./reports
  ○ CodeChecker parse ./reports -e html -o reports_html

● or 'store' online in webui/frontend
  ○ CodeChecker store ./reports --name mypkg@v0.9 \ --url http://localhost:8001/Default
<table>
<thead>
<tr>
<th>Diff</th>
<th>Name</th>
<th>Number of unresolved reports</th>
<th>Detection status</th>
<th>Analyzer statistics</th>
<th>Storage date</th>
<th>Analysis duration</th>
<th>Check command</th>
<th>Version tag</th>
<th>Description</th>
<th>CodeChecker version</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Diff Icon]</td>
<td>agl-service-gps@oneshot</td>
<td>1</td>
<td>![Red Circle Icon] (1)</td>
<td>clangsa: ✓ (1) &lt;br&gt; clang-tidy: ✓ (1)</td>
<td>2020-07-02 &lt;br&gt; 08:41:01</td>
<td>00:00:01</td>
<td>Show</td>
<td>![Hash Icon] dbf5618e00 &lt;br&gt;b2641197d8 &lt;br&gt;fa2f1599a37 &lt;br&gt;58909924</td>
<td>![Hash Icon]</td>
<td>6.13</td>
</tr>
<tr>
<td>![Diff Icon]</td>
<td>cynagora@oneshot</td>
<td>17</td>
<td>![Red Circle Icon] (17)</td>
<td>clang-tidy: ✓ (30) &lt;br&gt; clangsa: ✓ (30)</td>
<td>2020-07-02 &lt;br&gt; 08:00:16</td>
<td>00:00:35</td>
<td>Show</td>
<td>![Hash Icon] dbf5618e00 &lt;br&gt;b2641197d8 &lt;br&gt;fa2f1599a37 &lt;br&gt;58909924</td>
<td>![Hash Icon]</td>
<td>6.13</td>
</tr>
<tr>
<td>![Diff Icon]</td>
<td>app-framework-binder@oneshot</td>
<td>79</td>
<td>![Red Circle Icon] (79)</td>
<td>clangsa: ✓ (92) &lt;br&gt; ✓ (3) &lt;br&gt; clang-tidy: ✓ (2) &lt;br&gt; ✓ (3)</td>
<td>2020-07-02 &lt;br&gt; 07:50:44</td>
<td>00:02:04</td>
<td>Show</td>
<td>![Hash Icon] dbf5618e00 &lt;br&gt;b2641197d8 &lt;br&gt;fa2f1599a37 &lt;br&gt;58909924</td>
<td>![Hash Icon]</td>
<td>6.13</td>
</tr>
<tr>
<td>![Diff Icon]</td>
<td>app-framework-main@oneshot</td>
<td>35</td>
<td>![Red Circle Icon] (36)</td>
<td>clang-tidy: ✓ (34) &lt;br&gt; clangsa: ✓ (34)</td>
<td>2020-07-01 &lt;br&gt; 22:04:52</td>
<td>00:00:43</td>
<td>Show</td>
<td>![Hash Icon] dbf5618e00 &lt;br&gt;b2641197d8 &lt;br&gt;fa2f1599a37 &lt;br&gt;58909924</td>
<td>![Hash Icon]</td>
<td>6.13</td>
</tr>
<tr>
<td>![Diff Icon]</td>
<td>agl-service-audiomixer</td>
<td>4</td>
<td>![Red Circle Icon] (4)</td>
<td>clang-tidy: ✓ (2) &lt;br&gt; clangsa: ✓ (2)</td>
<td>2020-07-01 &lt;br&gt; 21:36:00</td>
<td>00:00:01</td>
<td>Show</td>
<td>![Hash Icon] dbf5618e00 &lt;br&gt;b2641197d8 &lt;br&gt;fa2f1599a37 &lt;br&gt;58909924</td>
<td>![Hash Icon]</td>
<td>6.13</td>
</tr>
</tbody>
</table>
Cool, I want that for my builds ...
Ok, I want CodeChecker for my OE/YP builds ...

What does the documentation say:

- [link](https://codechecker.readthedocs.io/en/latest/)
- There is a section about bitbake:
  - [link](https://codechecker.readthedocs.io/en/latest/analyzer/user_guide/#bitbake)
Do the following steps to log compiler calls made by BitBake using CodeChecker.

- Add `LD_LIBRARY_PATH`, `LD_PRELOAD`, `CC_LOGGER_GCC_LIKE` and `CC_LOGGER_FILE` to `BB_ENV_EXTRAWHITE` variable in your shell environment:

  ```bash
  export BB_ENV_EXTRAWHITE="LD_PRELOAD LD_LIBRARY_PATH CC_LOGGER_FILE CC_LOGGER_GCC_LIKE $BB_E
  ```

**Note:** `BB_ENV_EXTRAWHITE` specifies an additional set of variables to allow through (whitelist) from the external environment into BitBake's datastore.

- Add the following lines to the `conf/bitbake.conf` file:

  ```bash
  export LD_PRELOAD
  export LD_LIBRARY_PATH
  export CC_LOGGER_FILE
  export CC_LOGGER_GCC_LIKE
  ```

- Run `CodeChecker log`:

  ```bash
  CodeChecker log -o ..//compile_commands.json -b "bitbake myProject"
  ```
Hmmm ....

Rolling up sleeves:

Maybe a blind mouldwarp like I can do something about that!
meta-codechecker

● Integrates Codechecker seamlessly with bitbake
  ○ can write HTML reports
  ○ and upload to database
  ○ builds all necessary tools on-the-fly
    ■ requires meta-clang, meta-oe, meta-python

Where?: https://github.com/dl9pf/meta-codechecker
meta-codechecker - Example: step-by-step

git clone https://github.com/kraj/meta-clang.git

git clone https://git.openembedded.org/meta-openembedded

git clone https://github.com/dl9pf/meta-codechecker.git

# (check the meta-codechecker'S README.md)
git clone https://git.yoctoproject.org/git/poky
source poky/oe-init-build-env build-test-codechecker

bitbake-layers add-layer ../meta-clang
bitbake-layers add-layer ../meta-openembedded/meta-oe
bitbake-layers add-layer ../meta-openembedded/meta-python
bitbake-layers add-layer ../meta-codechecker

Next: edit conf/local.conf
meta-codechecker - Example: step-by-step

```bash
cat << EOF >> conf/local.conf
INHERIT += "codechecker"

# enable for all target packages:
CODECHECKER_ENABLED_class-target = "1"

# exempt clang
CODECHECKER_ENABLED_pn-clang = "0"

CODECHECKER_REPORT_HTML = "1"
EOF
```
meta-codechecker - Example: step-by-step

bitbake core-image-minimal
tree tmp/deploy/CodeChecker/
Summary

CodeChecker can be used by developers and in CI
complexity hidden by pre-loaded logger library
straightforward workflow
parsers into multiple formats
Webui to store and browse/review results
bitbake integration using meta-codechecker

documentation is good, but has a few dead links and such
Todo:

- add easy way to inject scanner configurations
  - e.g. select which issues to report (limit noise)
- deal with uploading report & password or token
- improve recipes using pipy currently
- layer vs CodeChecker UI on dockerhub container version
Call to action!

- Static Analysis can help improve your projects!
- Easy to use locally for development
- Integration to OpenEmbedded / Yocto Project
Thank you!

#dl9pf on freenode