Static Analysis with the Yocto Project

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

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

- 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.
- Possible integrations are presented
- 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'

cc1: 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 */
    ^
Cool, I want that for my builds ...
Allright, let's talk about clang and meta-clang !!

meta-clang is a layer that adds support for the Clang Compiler/Toolchain.

TLDR: software can be compiled with clang instead of gcc

Thus we can enable tooling like scan-build right away.

meta-clang

URL:  https://github.com/kraj/meta-clang

Main features:

- adds clang as compiler - selectively or
- clang for everything* (TOOLCHAIN = "clang")
- use compiler-rt as runtime (RUNTIME = "llvm")
- add clang to SDK (CLANGSDK = "1")
- enable scan-build (INHERIT += "scan-build")
Notes:

- Look at `meta-clang/conf/nonclangable.conf`
  - e.g. glibc, gcc, u-boot, grub
- `meta-clang/conf/nonscanable.conf`
meta-clang in action

git clone git://git.yoctoproject.org/poky

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

source poky/oe-init-build-env

bitbake-layers add-layer ..:/meta-clang
meta-clang in action (continued)

cat << EOF >> conf/site.conf

TOOLCHAIN = "clang"

CLANGSDK = "1"

EOF
meta-clang in action (continued)

bitbake core-image-minimal

option:

bitbake -c populage_sdk core-image-minimal
To enable scan-build do:

```
cat << EOF >> conf/site.conf
INHERIT += "scan-build"
SCAN_BUILD = ""
SCAN_BUILD_pn-busybox = "1"
CLANG_SCAN_SERVER_IP ??= "0.0.0.0"
EOF
```
meta-clang in action (continued)

To do a scan:

bitbake busybox

or

bitbake -c scanbuild busybox
meta-clang in action (continued)

To view the results:

```
bitbake -c scanview busybox
```

(currently broken?!)

alternative:

```
cd tmp/static-scan/busybox/*/ ; python3 -m http.server
```
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.
### Codechecker

**About**

CodeChecker is an analyzer tooling, defect database and viewer extension for the Clang Static Analyzer and Clang Tidy.

[Codechecker.readthedocs.io](https://codechecker.readthedocs.io)

**Tags**
- clang
- opp
- c
- clang-tidy
- static-analysis
- linux
- results-viewer
- macosx
- codechecker
- llvm
- analysis
- database
- objective-c
- defects
- docker
- static-analyzer
- static-analyzers

**Branch: master**

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>.github/ISSUE_TEMPLATE</td>
<td>[GitHub] Fix minor grammatical things in the issue templates</td>
<td>7 months ago</td>
</tr>
<tr>
<td>analyzer</td>
<td>[analyzer] Fix analyzer --file option</td>
<td>20 days ago</td>
</tr>
<tr>
<td>bin</td>
<td>[license] Change license (#2729)</td>
<td>last month</td>
</tr>
<tr>
<td>codechecker_common</td>
<td>Add a missing space in a debug warning</td>
<td>last month</td>
</tr>
<tr>
<td>config</td>
<td>Adding new checkers to the profiles, setting severities</td>
<td>2 months ago</td>
</tr>
<tr>
<td>docker</td>
<td>new dockerfiles for test environments</td>
<td>2 years ago</td>
</tr>
<tr>
<td>docs</td>
<td>[tools] tu_collector get dependent source files for headers</td>
<td>21 days ago</td>
</tr>
<tr>
<td>requirements_py/docs</td>
<td>Merge pull request #1935 from gyorb/readthedocs</td>
<td>15 months ago</td>
</tr>
<tr>
<td>scripts</td>
<td>[license] Change license (#2729)</td>
<td>last month</td>
</tr>
<tr>
<td>Name</td>
<td>Number of unresolved reports</td>
<td>Detection status</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>agl-service-gps@oneshot</td>
<td>1</td>
<td>🔴 (1)</td>
</tr>
<tr>
<td>cynagora@oneshot</td>
<td>17</td>
<td>🔴 (17)</td>
</tr>
<tr>
<td>app-framework-binder@oneshot</td>
<td>79</td>
<td>🔴 (79)</td>
</tr>
<tr>
<td>app-framework-main@oneshot</td>
<td>35</td>
<td>🔴 (36)</td>
</tr>
<tr>
<td>agl-service-audiomixer</td>
<td>4</td>
<td>🔴 (4)</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
Cool, I want that for my builds ...
Ok, I want CodeChecker for my OE/YP builds ...

What does the documentation say:

- There is a section about bitbake:
Do the following steps to log compiler calls made by BitBake using CodeChecker.

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

```
export BB_ENV_EXTRAWHITE="LD_PRELOAD LD_LIBRARY_PATH CC_LOGGER_FILE CC_LOGGER_GCCLIKE $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:

```
export LD_PRELOAD
export LD_LIBRARY_PATH
export CC_LOGGER_FILE
export CC_LOGGER_GCCLIKE
```

- Run `CodeChecker log`:

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

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

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

# disable all for now
CODECHECKER_ENABLED = "0"
# can enable _class-target for example!

# only busybox should use codechecker
CODECHECKER_ENABLED_pn-busybox = "1"

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

bitbake busybox

tree tmp/deploy/CodeChecker/
Summary meta-clang

+----------------------------------+
● meta-clang can be used by developers  
● CI use possible, but needs careful list of exemptions  
● straightforward workflow  
● bitbake integration

---------------------
● documentation is good  
● advanced use-cases need digging  
● scanview only per-package
Summary CodeChecker

++++++++++++++++++++

- 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 for meta-codechecker:

- 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 API vs CodeChecker UI dockerhub version
Call to action!

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

- meta-clang
- meta-codechecker
- github.com/Ericsson/CodeChecker

Not covered in this talk, but highly recommended

- meta-sca
Thank you!

#dl9pf on freenode
Hands-on Session
Part 1 - meta-clang (only)
Start a new ssh connection / terminal.

```
$>

source ~/yp-summit-may-21/poky/oe-init-build-env \
  ~/yp-summit-may-21/poky/build-analysis-clang

bitbake-layers add-layer ~/yp-summit-may-21/src/static/meta-clang/
```
meta-clang

$> rm conf/auto.conf
$> cat << EOF >> conf/auto.conf

# static analysis using clang
TOOLCHAIN = "clang"
CLANGSDK = "1"
INHERIT += "scan-build"
SCAN_BUILD = ""
SCAN_BUILD_pn-busybox = "1"
# report external server ip so we can view results ...
CLANG_SCAN_SERVER_IP = "$(lwp-request -o text checkip.dyndns.org | awk '{ print $NF }')"
EOF

$> bitbake -C unpack busybox
meta-clang

```bash
$> bitbake -c scanview busybox
```

Open URL printed in terminal and browse around.

Hit CTRL-C to end before we continue. (2 times if necessary)
Part 1 - meta-codechecker
Start a new ssh connection / terminal.

```bash
$>  source ~/yp-summit-may-21/poky/oe-init-build-env \
    ~/yp-summit-may-21/poky/build-analysis-codechecker

bitbake-layers add-layer ~/yp-summit-may-21/src/static/meta-clang/

bitbake-layers add-layer ~/yp-summit-may-21/src/static/meta-openembedded/meta-oe/

bitbake-layers add-layer ~/yp-summit-may-21/src/static/meta-openembedded/meta-python/

bitbake-layers add-layer ~/yp-summit-may-21/src/static/meta-codechecker/
```
```bash
$> rm conf/auto.conf
$> cat << EOF >> conf/auto.conf
  # static analysis using codechecker
  INHERIT += "codechecker"
  #e.g. enable for all target packages:
  #CODECHECKER_ENABLED_class-target = "1"
  # disable all
  CODECHECKER_ENABLED = "0"
  # only enable busybox
  CODECHECKER_ENABLED_pn-busybox = "1"
  # report into HTML files
  CODECHECKER_REPORT_HTML = "1"
EOF

bitbake busybox

  # check  tmp/deploy/CodeChecker/busybox/   !!
```
meta-codechecker

Step 2 ... upload to server for later inspection ...

```bash
$> cat << EOF >> conf/auto.conf
CODECHECKER_REPORT_STORE = "1"
CODECHECKER_REPORT_HOST = "http://ypdd.jsmo.de:8001/
CODECHECKER_REPORT_ENDPOINT = "$(hostname -s)-\${@'\${DATE}'.replace('..', '-')"
CODECHECKER_REPORT_ENDPOINT_CREATE = "1"
EOF
```

```bash
bitbake -C unpack busybox

# check [http://ypdd.jsmo.de:8001/](http://ypdd.jsmo.de:8001/) and locate the 'product' matching your hostname  !!
#
# e.g. devday0005-20210526
```
Then End. Thank you!