ELECTORSHIP SERIES



Static Analysis & Tools **Using Linux and Open Source Tools for Static Analysis** Dipl.-Ing. Jan-Simon Möller, AGL Release Manager, The Linux Foundation

ELF LIVE MENTORSHIP SERIES

Jan-Simon Möller

jsmoeller@linuxfoundation.org dl9pf@gmx.de, dl9pf on #freenode





Topics

- 1. What is 'Static Analysis'
- 2. Motivations to use it
- 3. Open Source Tools for Static Analysis
 - a. Example using gcc10
 - b. Example using clang
 - c. Example using cppcheck
 - d. Example using CodeChecker
- 4. Example integration in Makefiles
- 5. Example integration with git hooks
- 6. Summary
- 7. Q/A



What is 'Static Analysis'



- 'Static Analysis' or 'Static Code Analysis' in general is a method for debugging a program before it is run. It is done by analyzing the code in question and comparing it to a set of coding rules.
- This is in contrast to 'Dynamic Analysis' which means the program does run. (covered in an upcoming webinar)
- In most cases it is either performed on some parsed representation of the source code or on IR.



- Static analysis identifies defects before you run a program (e.g., between coding and unit testing).
- Dynamic analysis identifies defects when you run a program (e.g., during unit testing).
- However, some coding errors might not surface during unit testing. So, there are defects that dynamic testing might miss that static code analysis can find and of course vice versa.



- Usually it is performed by an automated tool.
- This type of analysis finds weaknesses and vulnerabilities.
- It is usually done early in the development cycle.



Motivations to use it



- Static analysis simply can find bugs early
- Static analysis can find hard-to-spot bugs
 - e.g. 30-level deep undefined/invalid access
- Static analysis can complement the peer review



- Static analysis is also used to comply with coding guidelines or industry standards
 - e.g. MISRA or ISO-26262
- It is also enforced for certain applications/industries:
 - o medical
 - o nuclear
 - o automotive, aviation



Open Source Tools for Static Analysis



• There are multiple categories of tools available

- tools using a form of string or pattern matching
- tools analyzing the code during compilation
- tools specialized for kernel space
- tools for userspace
- Of course there are also proprietary tools



The Linux Kernel is a very large and special codebase.

Currently it contains more than 20 million lines of code. This is very demanding on the tooling used. Thus there are specialized tools around the kernel:

- scripts/checkpatch.pl (string matching, basics&style, good for new submissions)
- sparse make C=1 CHECK="/usr/bin/sparse"
- coccinelle make C=1 CHECK="scripts/coccicheck" (see next tuesday's webinar by Julia Lawall)
- smatch make C=1 CHECK="smatch -p=kernel" (see webinar in ~2 weeky by Dan Carpenter)
- gcc / clang static analyser



For userspace there are a large number of tools available. A selection for C/C++ is below:

- gcc
- clang
- cppcheck
- coccinelle
- splint
- rats
- flawfinder









During development you can easily use these directly within your source tree:

- gcc (since gcc 10)
 - gcc -fanalyzer
- clang
 - o 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-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



```
> 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 */
```



```
> 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
                                                                                   //try: https://godbolt.org/
               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

//see: https://developers.redhat.com/blog/2020/03/26/static-analysis-in-gcc-10/



```
> 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 */
            ^{\sim}
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 */
```



> scan-build make

TLDR: replaces \$(CC) !!!

```
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-build: Run 'scan-view /tmp/scan-build-2020-10-15-161857-10509-1' to examine bug reports.
```

```
> 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)





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.

Ericsson / codechecker

⊙ Watch 49 ☆ Star 972

앟 Fork 111

<>Code () Issues 183 \$\% Pul	Il requests 35 🕞 Actions 🔟 Projects 3 🕕 Security 🗠 Insigh	nts				
양 Branch: master ◄	Go to file	Code - About				
gyorb committed dbf5618 7 days ago	 36 tags CodeChecker is an analyzer tooling, defect database and viewer extension for the Clang Static Analyzer and 					
.github/ISSUE_TEMPLATE	onths ago Clang Tidy					
analyzer 📄	analyzer [analyzer] Fix analyzerfile option 20 days ag					
📄 bin	[license] Change license (#2729) la	ast month clang cpp c clang-tidy				
codechecker_common	Add a missing space in a debug warning la	ast month static-analysis linux results-viewer				
Config	Adding new checkers to the profiles, setting severities 2 mo	onths ago				
ocker	new dockerfiles for test environments 2 y	years ago docker static-analyzer static-analyzers				
ocs docs	[tools] tu_collector get dependent source files for headers 21	days ago				
requirements_py/docs	Merge pull request #1935 from gyorb/readthedocs 15 mo	onths ago				
scripts	[license] Change license (#2729) la	ast month				



Image: Second state	cs 🖪 All reports	8	New features	х	Ξ	agl-service-gps@oneshot	х	
---	------------------	---	--------------	---	---	-------------------------	---	--

Search for runs...

Diff	Name	Number of unresolved reports	Detection status	Analyzer statistics	Storage date	Analysis duration	Check command	Version tag	Description	CodeCheck er version	Delete
00	agl-service-gps@oneshot	1	🌻 (1)	clangsa: ✔ (1) clang-tidy: ✔ (1)	2020-07-02 08:41:01	00:00:01	Show			6.13 (dbf5618c00 b26f41197d8 fa2f1599a37 58909924)	
00	cynagora@oneshot	17	4 (17)	clang-tidy: ✔ (30) clangsa: ✔ (30)	2020-07-02 08:00:16	00:00:35	Show			6.13 (dbf5618c00 b26f41197d8 fa2f1599a37 58909924)	
00	app-framework-binder@oneshot	79	\$ (79)	clangsa: ✓ (92) ¥ (3) clang-tidy: ✓ (92) ¥ (3)	2020-07-02 07:50:44	00:02:04	Show			6.13 (dbf5618c00 b26f41197d8 fa2f1599a37 58909924)	
00	app-framework-main@oneshot	35	🄹 (36)	clangsa: ✔ (34) clang-tidy: ✔ (34)	2020-07-01 22:04:52	00:00:43	Show			6.13 (dbf5618c00 b26f41197d8 fa2f1599a37 58909924)	
00	agl-service-audiomixer	4	尊 (4)	clang-tidy: ✔ (2) clangsa: ✔ (2)	2020-07-01 21:36:00	00:00:01	Show			6.13 (dbf5618c00 b26f41197d8 fa2f1599a37 58909924)	



Diff Delete



TLDR: does not need to change \$(CC)

Userspace tool CodeChecker is a set of python helpers

- main feature is that you wrap you 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



• 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

CodeChecker 6.12 Default

🕑 Runs 5 🔟 Checker statistics 🚍 All reports 💡 New features 🛪

Diff Delete

Diff	Name	Number of unresolved reports	Detection status	Analyzer statistics	Storage date	Analysis duration	Check command	Version tag	Description	CodeCheck er version	Delete
00	agl-service-gps@oneshot	1	🎄 (1)	 clangsa: ✓ (1) clang-tidy: ✓ (1) 	2020-07-02 08:41:01	00:00:01	Show			6.13 (dbf5618c00 b26f41197d8 fa2f1599a37 58909924)	
00	cynagora@oneshot	17	\$ (17)	• clang-tidy:	2020-07-02 08:00:16	00:00:35	Show			6.13 (dbf5618c00 b26f41197d8 fa2f1599a37 58909924)	
00	app-framework-binder@oneshot	79	🄹 (79)	• clangsa: (92) ★ (3) clang-tidy: (92) ★ (3) 	2020-07-02 07:50:44	00:02:04	Show			6.13 (dbf5618c00 b26f41197d8 fa2f1599a37 58909924)	
00	app-framework-main@oneshot	35	4 (36)	clangsa: ✔ (34) • clang-tidy: ✔ (34)	2020-07-01 22:04:52	00:00:43	Show			6.13 (dbf5618c00 b26f41197d8 fa2f1599a37 58909924)	
00	agl-service-audiomixer	4	(4)	 clang-tidy: ✓ (2) clangsa: ✓ (2) 	2020-07-01 21:36:00	00:00:01	Show			6.13 (dbf5618c00 b26f41197d8 fa2f1599a37 58909924)	

Back to product list 🥔 🔻

🛞 CodeChecker 6.12 Default







Example integration in Makefiles



- Integrating gcc's -fanalyzer into your Makefiles is easy: just add it to the CFLAGS !
- Similar for cmake . Add it to the CFLAGS.

```
# c source
TARGET EXEC ?= myprog
                                                    $(BUILD DIR)/%.c.o: %.c
BUILD DIR ?= ./build
                                                        $(MKDIR P) $(dir $@)
SRC DIRS ?= ./src
                                                        $(CC) $(CFLAGS) -c $< -o $@
SRCS := $(shell find $(SRC DIRS) -name *.c)
OBJS := $(SRCS:%=$(BUILD DIR)/%.0)
                                                    .PHONY: clean
DEPS := \$(OBJS:.o=.d)
INC DIRS := $(shell find $(SRC DIRS) -type d)
                                                    clean:
INC FLAGS := $(addprefix -I,$(INC DIRS))
                                                        $(RM) -r $(BUILD DIR)
CFLAGS ?= $(INC FLAGS) -Wall -Werror -fanalyzer
                                                    -include $(DEPS)
$(BUILD DIR)/$(TARGET EXEC): $(OBJS)
    $(CC) $(OBJS) -0 $@ $(LDFLAGS)
                                                    MKDIR P ?= mkdir -p
```



- If you use clang, you can run scan-build like so:
 <u>scan-build make <make options></u>
- It will add the flags on the fly.
 (If your Makefile uses \$(CC) !!)
- As shown, CodeChecker will record/reply the compilation without this need.



You can add cppcheck like so:

SOURCES = main.cpp

CPPCHECK = cppcheck

CHECKFLAGS = -q --error-exitcode=1

default: cppcheck.out.xml hellomake
.PHONY: default clean

cppcheck.out.xml: \$(SOURCES)
 \$(CPPCHECK) \$(CHECKFLAGS) \$^ -xml >\$@

hellomake: \$(OBJ)

\$(LINK.c) -o \$@ \$^



Example integration with git hooks



Git hooks are a mechanism that allows arbitrary code to be run before, or after, certain Git lifecycle events occur. For example, one could have a hook into the commit-msg event to validate that the commit message structure follows the <u>recommended format</u>.

The hooks can be any sort of executable code, including shell, PowerShell, Python, or any other scripts. Or they may be a binary executable. Anything goes! The only criteria is that hooks must be stored in the .git/hooks folder in the repo root, and that they must be named to match the corresponding events (as of Git 2.x):

TLF Live MENTORSHIP SERIES

- applypatch-msg
- pre-applypatch
- post-applypatch
- pre-commit
- prepare-commit-msg
- commit-msg
- post-commit
- pre-rebase

- post-checkout
- post-merge
- pre-receive
- update
- post-receive
- post-update
- post-rewrite
- pre-push





./myproject/.git/hooks/pre-commit

#!/bin/sh

echo "Running static analysis..."

Inspect code using scan-build, will exit 1 when bug is found scan-build make -j2

status=\$?

```
if [ "$status" = 0 ] ; then
      echo "Static analysis found no problems."
      exit 0
else
```

```
echo 1>&2 "Static analysis found violations."
exit 1
```



Example from:

https://github.com/danmar/cppcheck/blob/main/tools/git-pre-commit-cppcheck

[...]

```
# We should pass only added or modified C/C++ source files to cppcheck.
changed_files=$(git diff-index --cached $against | \
 grep -E '[MA] .*\.(c|cpp|cc|cxx)$' | cut -f 2)
```

```
if [ -n "$changed_files" ]; then
   cppcheck --error-exitcode=1 $changed_files
   exit $?
```

fi

exit O



Summary



- Static analysis
 - can help you improve your projects codebase early during coding
 - is one requirement in various standards / industries
 - can be easily added to your automation / CI



References:

https://github.com/dl9pf/staticanalysis-webinar

- <u>https://developers.redhat.com/blog/2020/03/26/static-analysis-in-gcc-10/</u>
- https://godbolt.org/
- https://clang.llvm.org/extra/clang-tidy/
- https://github.com/Ericsson/codechecker
- http://cppcheck.sourceforge.net/





ELF LIVE MENTORSHIP SERIES

Thank you for joining us today!

We hope it will be helpful in your journey to learning more about effective and productive participation in open source projects. We will leave you with a few additional resources for your continued learning:

- The <u>LF Mentoring Program</u> is designed to help new developers with necessary skills and resources to experiment, learn and contribute effectively to open source communities.
- <u>Outreachy remote internships program</u> supports diversity in open source and free software
- <u>Linux Foundation Training</u> offers a wide range of <u>free courses</u>, webinars, tutorials and publications to help you explore the open source technology landscape.
- <u>Linux Foundation Events</u> also provide educational content across a range of skill levels and topics, as well as the chance to meet others in the community, to collaborate, exchange ideas, expand job opportunities and more. You can find all events at <u>events.linuxfoundation.org</u>.