LF live

MENTORSHIP SERIES

LF Live MENTORSHIP SERIES

Static Analysis & Tools

Using Linux and Open Source Tools for Static Analysis

Dipl.-Ing. Jan-Simon Möller,
AGL Release Manager,
The Linux Foundation

LF 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'

LF Live MENTORSHIP SERIES

- '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:
  - medical
  - nuclear
  - automotive, aviation

LF Live MENTORSHIP SERIES

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

generic

↕

↕

security

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

```
> 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
          |
          |      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
```

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

//try: https://godbolt.org/

```
> 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 */
            ^
```

```
> 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)
```

```
1   #include <stddef.h>
2
3   int main(int argc, char *argv[]) {
4
5
6   int * pointer = NULL;
      1  'pointer' initialized to a null pointer value →
7   int value = *pointer; /* Dereferencing happens here */
      2  ← Dereference of null pointer (loaded from variable 'pointer')
8
9   return 0;
10
11  }
12
```

[https://github.com/Ericsson/**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.

Ericsson / **codechecker**

Watch 49 | Star 972 | Fork 111

<> **Code** | Issues 183 | Pull requests 35 | Actions | Projects 3 | Security | Insights

Branch: master | Go to file | Add file | Code

gyorb committed dbf5618 7 days ago ... ✕ | **3,818** commits | **5** branches | **36** tags

.github/ISSUE_TEMPLATE	[GitHub] Fix minor grammatical things in the issue templates	7 months ago
analyzer	[analyzer] Fix analyzer --file option	20 days ago
bin	[license] Change license (**#2729**)	last month
codechecker_common	Add a missing space in a debug warning	last month
config	Adding new checkers to the profiles, setting severities	2 months ago
docker	new dockerfiles for test environments	2 years ago
docs	[tools] tu_collector get dependent source files for headers	21 days ago
requirements_py/docs	Merge pull request **#1935** from gyorb/readthedocs	15 months ago
scripts	[license] Change license (**#2729**)	last month

About

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

codechecker.readthedocs.io

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

Readme

Apache-2.0 License

CodeChecker 6.12 Default

Back to product list

Runs 5 | Checker statistics | All reports | New features | agl-service-gps@oneshot

Search for runs...

Diff | Delete

Diff	Name	Number of unresolved reports	Detection status	Analyzer statistics	Storage date	Analysis duration	Check command	Version tag	Description	CodeChecker version	Delete
	agl-service-gps@oneshot	1	(1)	clangsa: ✔ (1) clang-tidy: ✔ (1)	2020-07-02 08:41:01	00:00:01	Show			6.13 (dbf5618c00b26f41197d8fa2f1599a3758909924)	
	cynagora@oneshot	17	(17)	clang-tidy: ✔ (30) clangsa: ✔ (30)	2020-07-02 08:00:16	00:00:35	Show			6.13 (dbf5618c00b26f41197d8fa2f1599a3758909924)	
	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 (dbf5618c00b26f41197d8fa2f1599a3758909924)	
	app-framework-main@oneshot	35	(36)	clangsa: ✔ (34) clang-tidy: ✔ (34)	2020-07-01 22:04:52	00:00:43	Show			6.13 (dbf5618c00b26f41197d8fa2f1599a3758909924)	
	agl-service-audiomixer	4	(4)	clang-tidy: ✔ (2) clangsa: ✔ (2)	2020-07-01 21:36:00	00:00:01	Show			6.13 (dbf5618c00b26f41197d8fa2f1599a3758909924)	

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

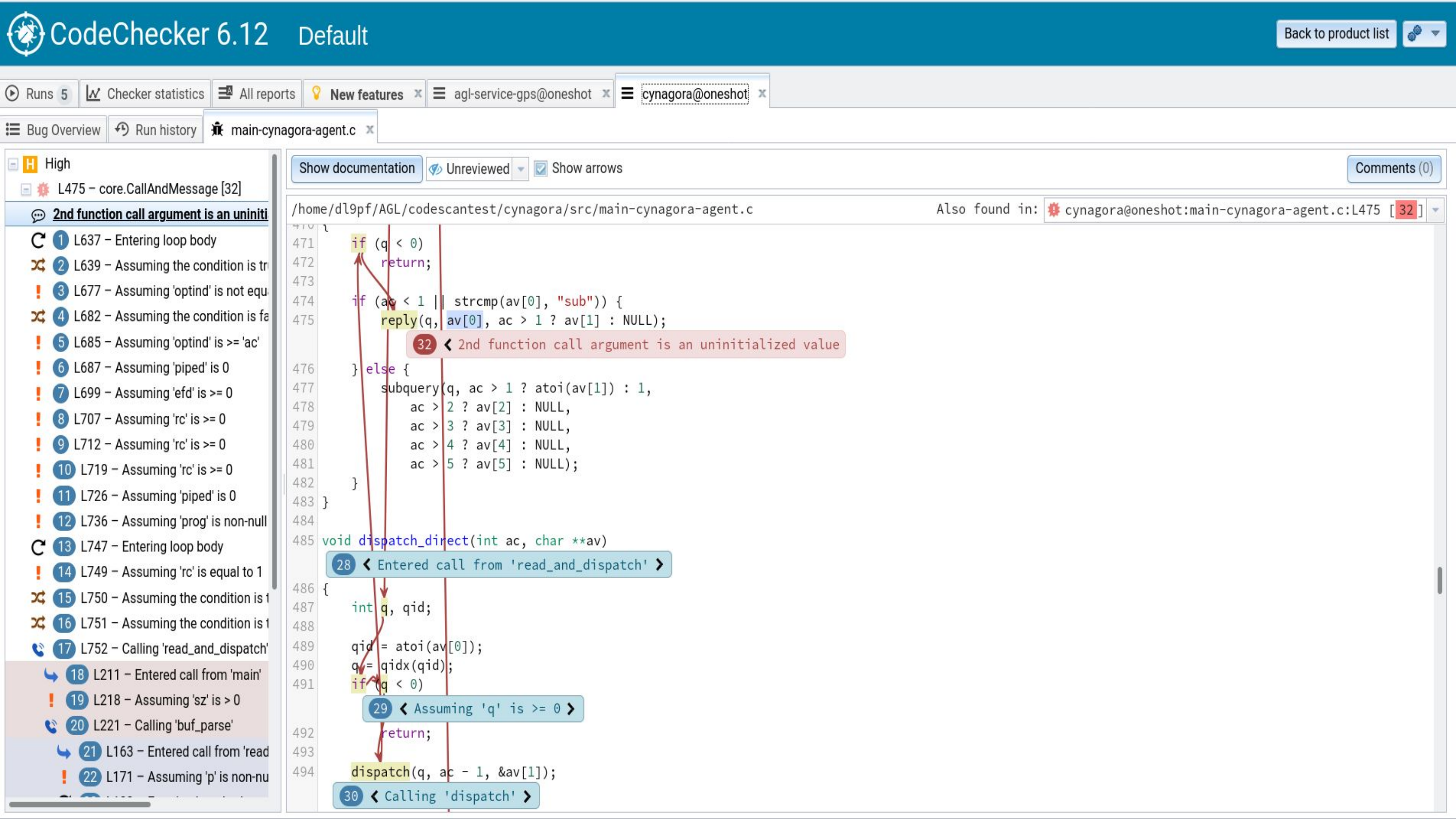
Back to product list

Runs 5 | Checker statistics | All reports | New features

Search for runs...

Diff Delete

Diff	Name	Number of unresolved reports	Detection status	Analyzer statistics	Storage date	Analysis duration	Check command	Version tag	Description	CodeChecker version	Delete
	agl-service-gps@oneshot	1	(1)	clangsa: ✔ (1) clang-tidy: ✔ (1)	2020-07-02 08:41:01	00:00:01	Show			6.13 (dbf5618c00b26f41197d8fa2f1599a3758909924)	
	cynagora@oneshot	17	(17)	clang-tidy: ✔ (30) clangsa: ✔ (30)	2020-07-02 08:00:16	00:00:35	Show			6.13 (dbf5618c00b26f41197d8fa2f1599a3758909924)	
	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 (dbf5618c00b26f41197d8fa2f1599a3758909924)	
	app-framework-main@oneshot	35	(36)	clangsa: ✔ (34) clang-tidy: ✔ (34)	2020-07-01 22:04:52	00:00:43	Show			6.13 (dbf5618c00b26f41197d8fa2f1599a3758909924)	
	agl-service-audiomixer	4	(4)	clang-tidy: ✔ (2) clangsa: ✔ (2)	2020-07-01 21:36:00	00:00:01	Show			6.13 (dbf5618c00b26f41197d8fa2f1599a3758909924)	


CodeChecker 6.12
Default
Back to product list
Runs 5
Checker statistics
All reports
New features
agl-service-gps@oneshot
cynagora@oneshot
Bug Overview
Run history
main-cynagora-agent.c
High
L475 – core.CallAndMessage [32]
2nd function call argument is an uniniti
1 L637 – Entering loop body
2 L639 – Assuming the condition is tr
3 L677 – Assuming 'optind' is not equ
4 L682 – Assuming the condition is fa
5 L685 – Assuming 'optind' is >= 'ac'
6 L687 – Assuming 'piped' is 0
7 L699 – Assuming 'efd' is >= 0
8 L707 – Assuming 'rc' is >= 0
9 L712 – Assuming 'rc' is >= 0
10 L719 – Assuming 'rc' is >= 0
11 L726 – Assuming 'piped' is 0
12 L736 – Assuming 'prog' is non-null
13 L747 – Entering loop body
14 L749 – Assuming 'rc' is equal to 1
15 L750 – Assuming the condition is t
16 L751 – Assuming the condition is t
17 L752 – Calling 'read_and_dispatch'
18 L211 – Entered call from 'main'
19 L218 – Assuming 'sz' is > 0
20 L221 – Calling 'buf_parse'
21 L163 – Entered call from 'read
22 L171 – Assuming 'p' is non-nu
Show documentation
Unreviewed
Show arrows
Comments (0)
/home/dl9pf/AGL/codescantest/cynagora/src/main-cynagora-agent.c
Also found in: cynagora@oneshot:main-cynagora-agent.c:L475 [32]
if (q < 0)
return;
if (ac < 1 || strcmp(av[0], "sub")) {
reply(q, av[0], ac > 1 ? av[1] : NULL);
32 2nd function call argument is an uninitialized value
} else {
subquery(q, ac > 1 ? atoi(av[1]) : 1,
ac > 2 ? av[2] : NULL,
ac > 3 ? av[3] : NULL,
ac > 4 ? av[4] : NULL,
ac > 5 ? av[5] : NULL);
}
}
void dispatch_direct(int ac, char **av)
28 Entered call from 'read_and_dispatch'
{
int q, qid;
qid = atoi(av[0]);
q = qidx(qid);
if (q < 0)
29 Assuming 'q' is >= 0
return;
dispatch(q, ac - 1, &av[1]);
30 Calling 'dispatch'

LF Live MENTORSHIP SERIES

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.

```
TARGET_EXEC ?= myprog
BUILD_DIR ?= ./build
SRC_DIRS ?= ./src

SRCS := $(shell find $(SRC_DIRS) -name *.c)
OBJS := $(SRCS:%=$(BUILD_DIR)/%.o)
DEPS := $(OBJS:.o=.d)
INC_DIRS := $(shell find $(SRC_DIRS) -type d)
INC_FLAGS := $(addprefix -I,$(INC_DIRS))
CFLAGS ?= $(INC_FLAGS) -Wall -Werror -fanalyzer

$(BUILD_DIR)/$(TARGET_EXEC): $(OBJS)
    $(CC) $(OBJS) -o $@ $(LDFLAGS)
```

```
# c source
$(BUILD_DIR)/%.c.o: %.c
    $(MKDIR_P) $(dir $@)
    $(CC) $(CFLAGS) -c $< -o $@

.PHONY: clean

clean:
    $(RM) -r $(BUILD_DIR)

-include $(DEPS)

MKDIR_P ?= mkdir -p
```

- If you use clang, you can run scan-build like so:
  - **scan-build make <make options>**
- 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 $@ $^
```

LF Live MENTORSHIP SERIES

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 recommended format.

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):

LF 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
fi
```

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 0
```

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

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

LF live MENTORSHIP SERIES

Q/A

LF 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 LF Mentoring Program is designed to help new developers with necessary skills and resources to experiment, learn and contribute effectively to open source communities.
- Outreachy remote internships program supports diversity in open source and free software
- Linux Foundation Training offers a wide range of free courses, webinars, tutorials and publications to help you explore the open source technology landscape.
- Linux Foundation Events 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 events.linuxfoundation.org.