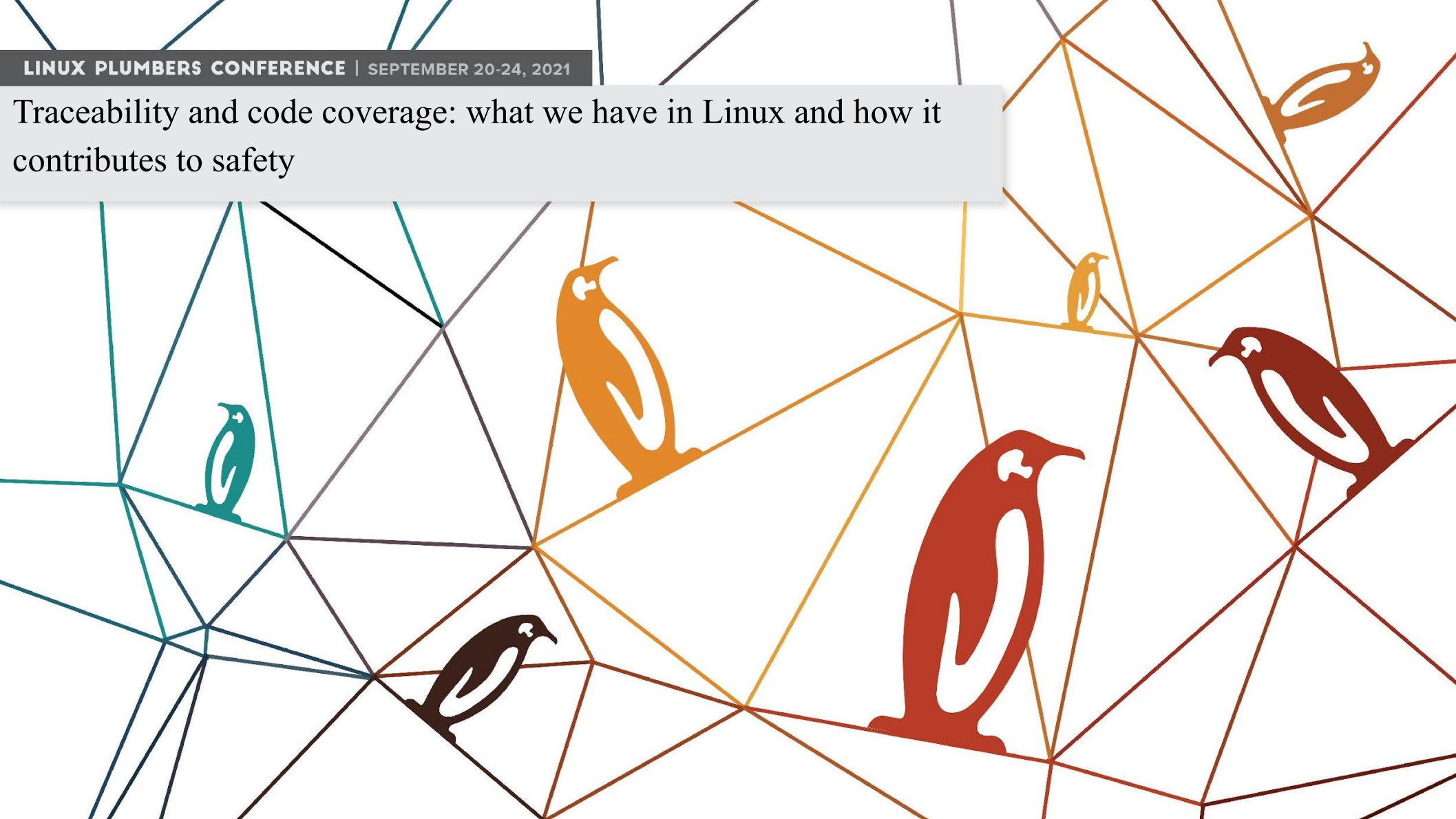
LINUX PLUMBERS CONFERENCE | SEPTEMBER 20-24, 2021

Traceability and code coverage: what we have in Linux and how it contributes to safety



Presenter

Rachel Sibley
Senior Principal Quality Engineer
Functional Safety
In-vehicle OS

Red Hat

Agenda:

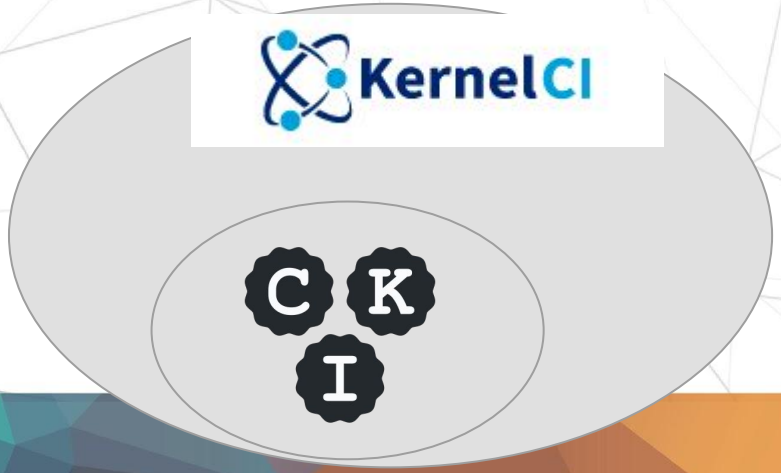
- Overview of Red Hat Automotive Initiative
- Overview of the Kernel CI and CKI project
- Importance of Code Coverage in Functional Safety Systems
- Code Coverage Analysis
- Targeted Testing
- Possible Improvements

Overview of Red Hat Automotive Initiative

- RH plans to deliver first continuously certified Linux OS for Road Vehicles
- Based on RHEL
- New areas to explore including Infotainment and Functional Safety
- Leverage existing testing and tooling we already use for RHEL
- Functional Safety will drive the requirements and tasks to close potential gaps/hazards

Overview of CKI and Kernel CI

- CKI is Red Hat's public facing CI system responsible for running gating tests for upstream/RHEL kernels
  - Provide testing & feedback for upstream kernel trees (mainline, stable, subsystems ...)
- Member of the Kernel CI upstream project and Linux Foundation https://kernelci.org/
- CKI is one of many contributors to KCIDB (Kernel CI Database), including Kernel CI
- Both Targeted testing and Code Coverage analysis are priority features for Test coverage for CKI



Importance of Code Coverage in Functional Safety Systems

ISO26262-6-9.4.4 states "*To evaluate the completeness of verification and to provide evidence that the objectives for unit testing are adequately achieved, the coverage of requirements at the software unit level shall be determined and the structural coverage shall be measured in accordance with the metrics as listed in Table 9. If the achieved structural coverage is considered insufficient, either additional test cases shall be specified or a rationale based on other methods shall be provided.*"

So Code Coverage is needed to make sure that we tested all the code that is supporting a safety claim; the rigor of the coverage technique increases according to the ASIL Level.

Moreover code coverage helps to verify the correct traceability of the requirements down to the design. I.e. if running a requirement based test shows coverage of code belonging to a SW module that was not in the traceability scope, it means that something was missed

Coverage technique	**ASIL Level**			
	A	B	C	D
Statement Coverage	++	++	+	+
Branch Coverage	+	++	++	++
MC/DC Coverage	+	+	+	++

Code Coverage Analysis

- What we have today for CKI
  - Manually triggered by a bot
  - Fetches a pre-built RHEL gcov kernel
  - CI results are linked back to the Merge Request
  - Work is ongoing to enable it in the CI pipeline and work with upstream kernels
- Uses lcov to capture and generate the html coverage reports
- Possible to merge multiple test reports
- KDIR param can be used to provide more granular reports
- Code coverage tests are located in CKI's open source test repo kernel-tests

Code Coverage Analysis

Example report for running xfstests

- --kdir KDIR Capture kernel coverage data only from this subdirectory
- `<param name="KDIR" value="fs/xfs"/>`

LCOV - code coverage report

Current view:	top level			
Test:	Coverage of kernel tests on 4.18.0-161.el8.gcov.x86_64			
Date:	2019-12-11 19:56:23			
Legend:	Rating: low: < 75 % / medium: >= 75 % / high: >= 90 %			
		Hit	Total	Coverage
Lines:		33143	44787	74.0 %
Functions:		2171	3411	63.6 %

Directory	Line Coverage		Functions	
arch/x86/include/asm	93.3 %	56 / 60	100.0 %	1 / 1
fs/xfs	73.3 %	17077 / 23298	60.7 %	1390 / 2289
fs/xfs/libxfs	75.1 %	15418 / 20542	72.8 %	756 / 1038
include/asm-generic	81.2 %	26 / 32	0.0 %	0 / 1
include/linux	65.5 %	544 / 831	29.3 %	24 / 82
include/linux/byteorder	100.0 %	3 / 3	-	0 / 0
include/linux/sched	100.0 %	7 / 7	-	0 / 0
include/linux/unaligned	75.0 %	6 / 8	-	0 / 0
include/uapi/linux	100.0 %	3 / 3	-	0 / 0
include/uapi/linux/byteorder	100.0 %	3 / 3	-	0 / 0

Generated by: LCOV version 1.14-2-gaa56a43

Targeted Testing

- CKI provides targeted testing for Patches, upstream trees, and drivers
  - Patches: regex patterns in changed patches sources
  - Upstream: Targeted testing for upstream kernel subsystem trees, e.g run networking related tests on net-next
  - Drivers: Target specific drivers and hardware for testing
- CLI for generating xml to be used by our provisioner based on set, tree, arch ..
  - kpet: https://gitlab.com/CKI-project/kpet
- Database which store config files to tell Pipeline where/how to test (kpet-db)
  - kpet-db: https://gitlab.com/CKI-project/kpet-db
- Plan to enable gating for patches which include no targeted test coverage
- Examples:
  - To preview patch generated test cases:
    - kpet run test list 001.patch
  - To generate complete Beaker XML:
    - kpet run generate -a aarch64 -k '##KPG_URL##' -t upstream 001.patch

Possible Improvements

Future Improvements for CKI include:

- Gating patches with missing targeted tests
- Integrating code coverage analysis into the pipeline

Questions :

1. Who else is actively using code coverage analysis ? How do you use it ? Do you find it useful ?
2. What is the % coverage should we target, is >= 90 % code coverage realistic ?
3. Building gcov kernels can be time consuming and expensive, how often should we run/generate reports ?
4. If not running code coverage analysis for every patch, how do we ensure coverage is good enough ?
5. Besides code coverage analysis and targeted testing, what other methods are recommended for measuring code coverage ?
6. Code Coverage as verification measure of expected traceability of requirements down to the design. What are we missing so far in Linux?