Linux Kernel Testing: Where Are We ?

Guenter Roeck, Google
linux@roeck-us.net

Agenda

- Test Suites
- Testbeds
- Summary
- Next steps

Test Suites

Test Suites

- Linux Test Project (LTP)
- Module tests in tools/testing
  - kselftest
  - nvdimm
  - ...
- Static code analyzers
- Fuzzing tools
- Subsystem tests
  - e.g. xfstests

Linux Test Project (LTP)

- Collection of tools for testing the Linux kernel and related features
- Started by SGI
- Maintained by IBM, Cisco, Fujitsu, SUSE, Red Hat and others

LTP - Continued

- Coverage
  - 1000+ system calls
  - 1000+ POSIX conformance tests
  - 400+ IO stress tests
  - Realtime, networking, cgroups, namespace tests
- Links
  - https://linux-test-project.github.io/
  - https://github.com/linux-test-project/ltp/wiki

Kernel self-test

- Unit test framework in Linux kernel
- Driven by Shuah Khan
- Part of Linux kernel source
- Links
    - tools/testing/selftests/
    - https://kselftest.wiki.kernel.org/
    - https://lwn.net/Articles/608959/

Fuzzing Tools

- Trinity
  - Maintained by Dave Jones
  - A Linux System call fuzz tester
  - http://codemonkey.org.uk/projects/trinity/
  - https://github.com/kernelslacker/trinity
- Syzcaller
  - Developed and maintained by the Google syzcaller team
  - Unsupervised, coverage-guided Linux syscall fuzzer
  - Meant to be used with KASAN
  - https://github.com/google/syzkaller

Static Code Analyzers

- Coccinelle
  - Developed and maintained by Julia Lawall
  - A program matching and transformation engine
  - http://coccinelle.lip6.fr/
- Coverity
  - Commercial Static Analyzer
  - Linux kernel tested for free
  - Detailed test results and statistics available

Static Code Analyzers - Continued

- gcc warnings
- smatch
  - “The Source Matcher”
  - https://blogs.oracle.com/linuxkernel/entry/smatch_static_analysis_tool_overview
- sparse
  - A Semantic Parser for C
  - https://sparse.wiki.kernel.org/index.php/Main_Page

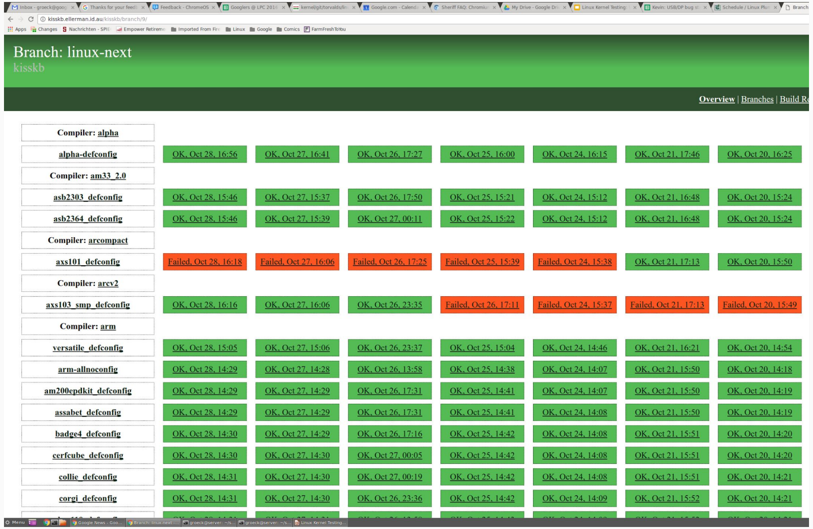
Automated Testing

Automated Testing

- Autobuilders
  - kisskb
  - 0Day
  - kernelci.org
  - Kerneltests.org
  - Other
    - Olof’s autobuilder, autobooter
    - Tegra builds
    - Buildbot for Mark Brown
- Static Analysis
  - Coverity

Kisskb

- Set up and maintained by Michael Ellerman
- The ‘original’ automated kernel build system
- Online (at least) since 2007
- Coverage
    - Most architectures (29)
    - Mainline, next, stable
    - Build only, no boot/runtime tests



Branch: linux-next

kisskb

Overview | Branches | Build Re

Compiler: alpha							
alpha-defconfig	OK, Oct 28, 16:56	OK, Oct 27, 16:41	OK, Oct 26, 17:27	OK, Oct 25, 16:00	OK, Oct 24, 16:15	OK, Oct 21, 17:46	OK, Oct 20, 16:25
Compiler: am33_2.0							
asb2303_defconfig	OK, Oct 28, 15:46	OK, Oct 27, 15:37	OK, Oct 26, 17:50	OK, Oct 25, 15:21	OK, Oct 24, 15:12	OK, Oct 21, 16:48	OK, Oct 20, 15:24
asb2364_defconfig	OK, Oct 28, 15:46	OK, Oct 27, 15:39	OK, Oct 27, 00:11	OK, Oct 25, 15:22	OK, Oct 24, 15:12	OK, Oct 21, 16:48	OK, Oct 20, 15:24
Compiler: arcompact							
axs101_defconfig	Failed, Oct 28, 16:18	Failed, Oct 27, 16:06	Failed, Oct 26, 17:25	Failed, Oct 25, 15:39	Failed, Oct 24, 15:38	OK, Oct 21, 17:13	OK, Oct 20, 15:50
Compiler: arcv2							
axs103_smp_defconfig	OK, Oct 28, 16:16	OK, Oct 27, 16:06	OK, Oct 26, 23:35	Failed, Oct 26, 17:11	Failed, Oct 24, 15:37	Failed, Oct 21, 17:13	Failed, Oct 20, 15:49
Compiler: arm							
versatile_defconfig	OK, Oct 28, 15:05	OK, Oct 27, 15:06	OK, Oct 26, 23:37	OK, Oct 25, 15:04	OK, Oct 24, 14:46	OK, Oct 21, 16:21	OK, Oct 20, 14:54
arm-allnoconfig	OK, Oct 28, 14:29	OK, Oct 27, 14:28	OK, Oct 26, 13:58	OK, Oct 25, 14:38	OK, Oct 24, 14:07	OK, Oct 21, 15:50	OK, Oct 20, 14:18
am200epdkit_defconfig	OK, Oct 28, 14:29	OK, Oct 27, 14:29	OK, Oct 26, 17:31	OK, Oct 25, 14:41	OK, Oct 24, 14:07	OK, Oct 21, 15:50	OK, Oct 20, 14:19
assabet_defconfig	OK, Oct 28, 14:29	OK, Oct 27, 14:29	OK, Oct 26, 17:31	OK, Oct 25, 14:41	OK, Oct 24, 14:08	OK, Oct 21, 15:50	OK, Oct 20, 14:19
badge4_defconfig	OK, Oct 28, 14:30	OK, Oct 27, 14:29	OK, Oct 26, 17:16	OK, Oct 25, 14:42	OK, Oct 24, 14:08	OK, Oct 21, 15:51	OK, Oct 20, 14:20
cerfcube_defconfig	OK, Oct 28, 14:30	OK, Oct 27, 14:30	OK, Oct 27, 00:05	OK, Oct 25, 14:42	OK, Oct 24, 14:08	OK, Oct 21, 15:51	OK, Oct 20, 14:20
collie_defconfig	OK, Oct 28, 14:31	OK, Oct 27, 14:30	OK, Oct 27, 00:19	OK, Oct 25, 14:42	OK, Oct 24, 14:08	OK, Oct 21, 15:51	OK, Oct 20, 14:21
corgi_defconfig	OK, Oct 28, 14:31	OK, Oct 27, 14:30	OK, Oct 26, 23:36	OK, Oct 25, 14:42	OK, Oct 24, 14:09	OK, Oct 21, 15:52	OK, Oct 20, 14:21

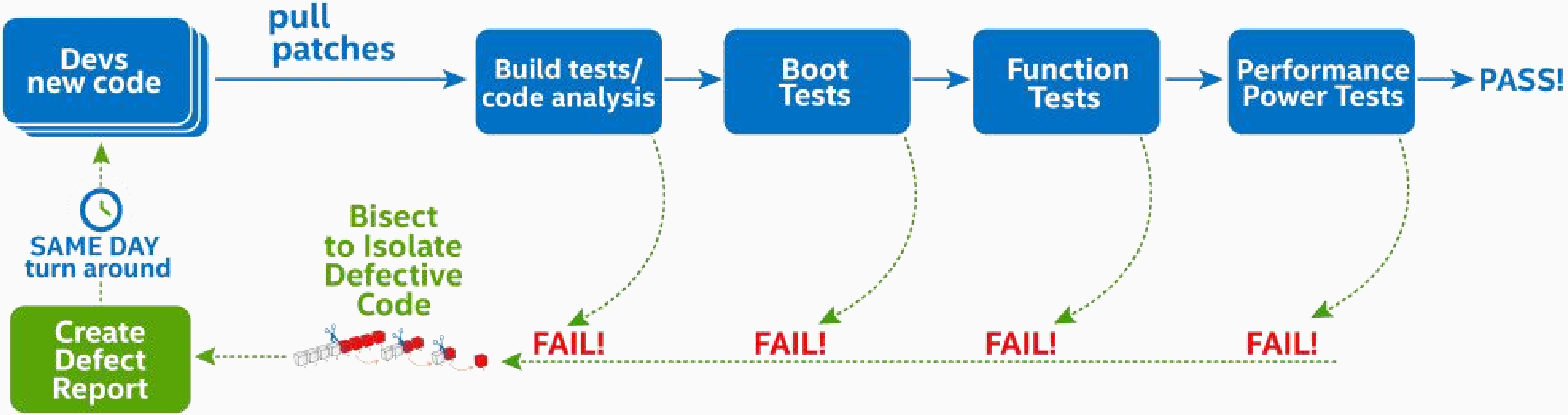
Kisskb - Continued

- Used to generate weekly “Build regressions/improvements…” reports
- Build results available per e-mail on request
- Links
  - http://kisskb.ellerman.id.au/kisskb/matrix

0Day: Overview

- Fengguang Wu’s brain child
- Operational since 2013
- Finds and reports
  - Build failures
  - Boot failures
  - Functional bugs
  - Performance regressions and improvements
- By far the most comprehensive test bed

0Day: How does it work ?



Devs new code

pull patches

Build tests/ code analysis

Boot Tests

Function Tests

Performance Power Tests

PASS!

FAIL!

FAIL!

FAIL!

FAIL!

Bisect to Isolate Defective Code

SAME DAY turn around

Create Defect Report

0Day: Infrastructure

- ~80 servers
  - 18 build servers
  - Other servers used for runtime tests
- ~8 engineers

0Day: Coverage

- 683 Trees
  - Mainline, stable, stable-rc, next
  - Developer trees
- Detects ~1,200 daily branch changes
- Supports almost all kernel architectures
  - Exceptions: metag, arc, hexagon, unicore32
- Up to 2,000 test cases

0Day: Statistics

- 36,000 builds per day
- 150,000 runtime tests per day
    - ~8000 functional / performance / power tests
    - Remaining tests are boot/trinity tests in qemu
- ~800 build errors reported per month
- ~60 qemu boot failures reported per month
- 60% of failures reported within 2 hours
- 90% of failures reported within 24 hours
- Boot tests may require up to 1 week to complete
- Performance tests may require up to 1 month to complete

0Day: Challenges (from Fengguang)

- UI needs to be improved
- Runtime tests are noisy
  - Reporting delays (long runtime, system load)
  - Difficult to reproduce
  - Difficult to interpret
- High maintenance burden
  - Bugs, noisiness
  - Keeps the entire team busy

0Day: Links

- https://01.org/lkp
- https://git.kernel.org/cgit/linux/kernel/git/wfg/lkp-tests.git
- https://lists.01.org/pipermail/kbuild-all

Kernelci: Overview

- Maintained by Kevin Hillman
- Operational since May 2014
- Goals
  - Wide range of Hardware
  - Quickly find regressions
  - Distributed
    - 9 board farms, with more coming
  - Framework independent
    - Most farms use Linaro LAVA

Kernelci: Coverage

- Mainline, next, arm-soc
- Stable, stable release candidates
- Various maintainer trees
- Arm, arm64, x86, mips
- All upstream default configurations, plus variants
  - 260+ configurations
- Build and boot; no runtime tests, no bisect (yet)
- Summary reports for stable release candidates

Kernelci: Statistics

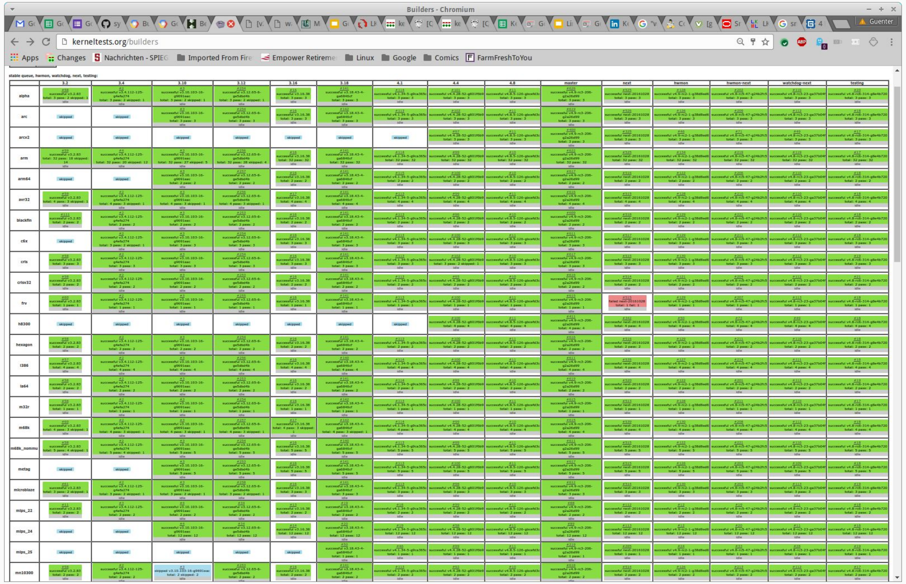
- 9 Build farms
- 31 unique SoCs (arm, arm64, x86, MIPS)
- 260+ Configurations
- 200+ Unique boards
- 1000+ Builds per day
- 2300+ boots per day

Kernelci: Links

- https://kernelci.org/
- https://lists.linaro.org/pipermail/kernel-build-reports/
- #kernelci on IRC, Freenode

Kerneltests: Overview

- Created to test stable release candidates
- Operational since 2013
- Goals
  - Build all architectures
    - Reasonable snapshot of default configurations
  - Boot all available qemu emulations
  - Basic runtime tests (to be added …)
- Runs on five PCs with i7 class CPUs



Builders - Chromium

kerneltests.org/builders

stable queue, hwmon, watchdog, next, testing:

3.2 | 3.4 | 3.10 | 3.12 | 3.16 | 3.18 | 4.1 | 4.4 | 4.8 | master | next | hwmon | hwmon-next | watchdog-next | testing

alpha
arc
arcv2
arm
arm64
avr32
blackfin
c6x
cris
crisv32
frv
h8300
hexagon
i386
ia64
m32r
m68k
m68k_nommu
metag
microblaze
mips_22
mips_24
mips_25
mn10300

Kerneltests: Coverage

- Branches
  - stable-rc, mainline, next, hwmon, watchdog
- Builds
  - All architectures and variants
  - Up to 149 defconfigs
- Boot tests (qemu)
  - 14 architectures (+variants)
  - Up to 113 platforms
- Summary reports for stable release candidates
- No runtime tests, no bisect, no individual reports

Kerneltests: Statistics

- Builds
  - 15 branches
  - Up to 149 builds per branch
  - 39 architectures and architecture variants
- Qemu tests
  - 14 architectures, 8 variants (little/big endian, 32/64 bit)
  - Up to 113 platform boots per branch
- Average 300-400 builds, 200-300 boots per day

Kerneltests: Challenges

- UI
- Buildbot stability
- No long-term storage of test results
- Automated reports
- Automated bisect
- Maintenance
  - Toolchains
  - Qemu
- Operational cost

Kerneltests: Links

- http://kerneltests.org/builders
- https://github.com/groeck/linux-build-test
- https://github.com/groeck/qemu

Other Build and Test Systems

- Mark Brown’s Buildbot
  - x86_64, arm, arm64 (8 builds)
- Olof's Autobuilder
  - mainline and next for arm, arm64, powerpc
  - ~120 configurations
- Olof’s Autobooter
  - mainline, next, arm-soc
  - ~75 boards (arm)
- Tegra Builds
  - Various Tegra builds and boots on mainline

Results reported at https://lists.linaro.org/pipermail/kernel-build-reports/

Coverity

- Detailed static analysis on Linux kernel
- Detailed defect reports and statistics
- Kernel contributors get free account to see results
- https://scan.coverity.com/projects/linux

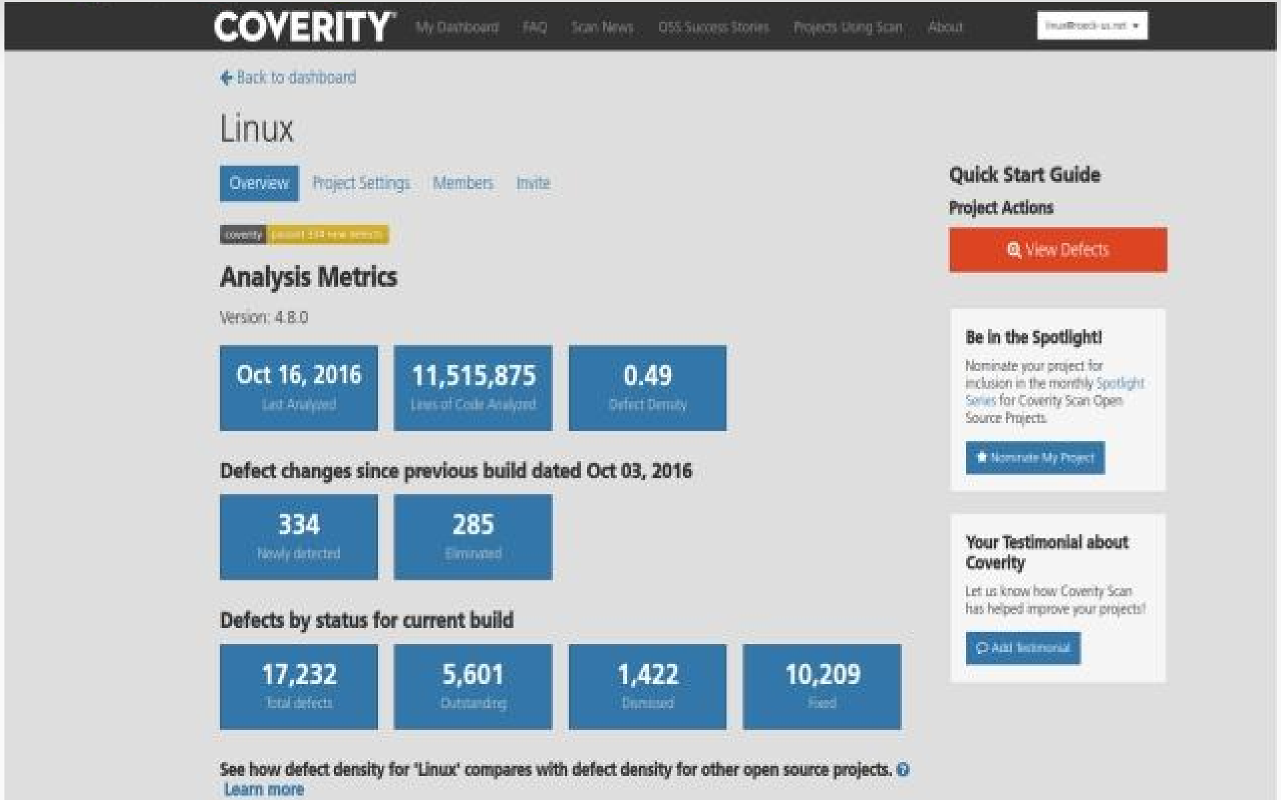
Coverity: Sample report

```
*** CID 1374326:  Incorrect expression  (NO_EFFECT)
/tools/objtool/arch/x86/decode.c: 102 in arch_decode_instruction()
96                  insn.modrm.nbytes && insn.modrm.bytes[0] == 0xe5)
97                          /* mov rsp, rbp */
98                          *type = INSN_FP_SETUP;
99              break;
100
101     case 0x8d:

  CID 1374326:  Incorrect expression  (NO_EFFECT)
   Comparing an array to null is not useful: "insn.rex_prefix.bytes", since the test will always evaluate as true.

102             if (insn.rex_prefix.bytes &&                          /* Should probably be insn.rex_prefix.nbytes */
103                 insn.rex_prefix.bytes[0] == 0x48 &&
104                 insn.modrm.nbytes && insn.modrm.bytes[0] == 0x2c &&
105                 insn.sib.nbytes && insn.sib.bytes[0] == 0x24)
106                         /* lea %(rsp), %rbp */
107                         *type = INSN_FP_SETUP
```

Coverity: Statistics



Linux

Analysis Metrics

Version: 4.8.0

Oct 16, 2016	11,515,875	0.49
Last Analyzed	Lines of Code Analyzed	Defect Density

Defect changes since previous build dated Oct 03, 2016

334	285
Newly detected	Eliminated

Defects by status for current build

17,232	5,601	1,422	10,209
Total defects	Outstanding	Dismissed	Fixed

See how defect density for 'Linux' compares with defect density for other open source projects.
Learn more

Quick Start Guide

Project Actions

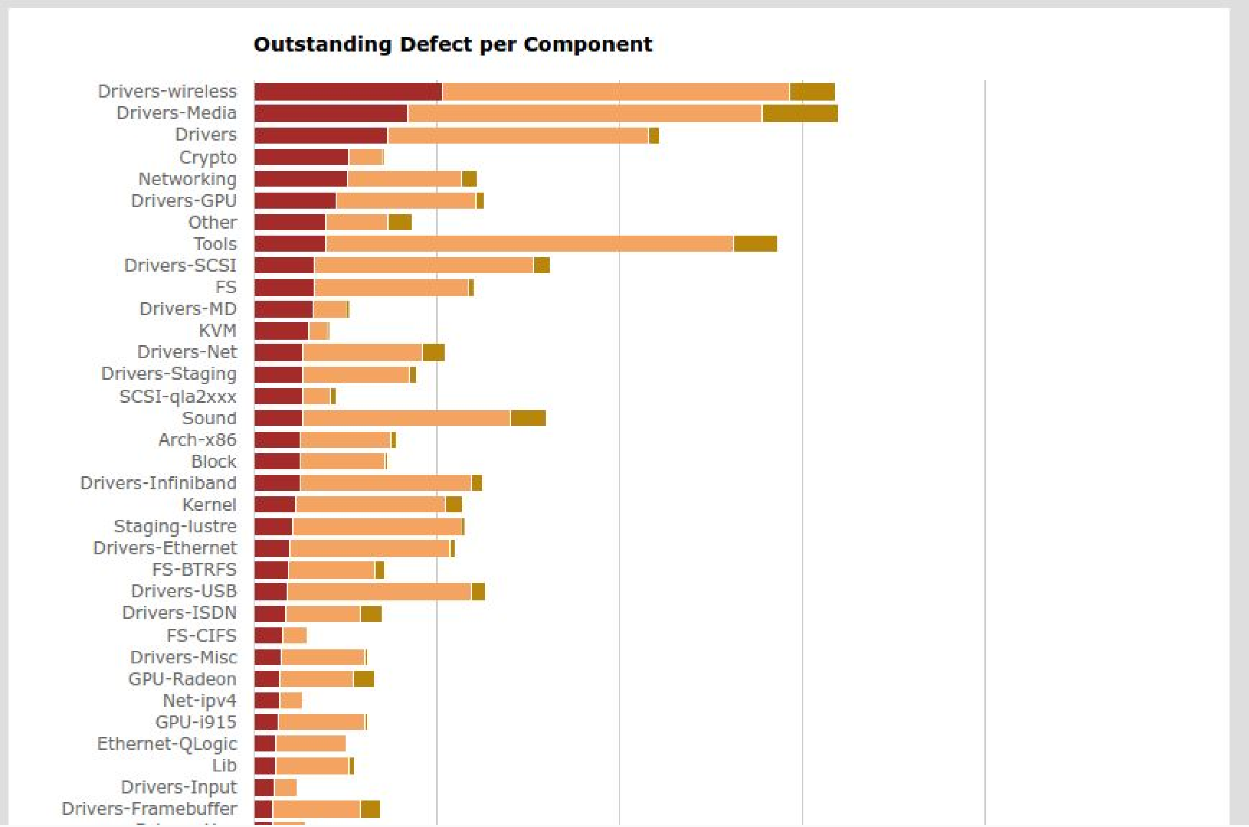
Be in the Spotlight!

Nominate your project for inclusion in the monthly Spotlight Series for Coverity Scan Open Source Projects.

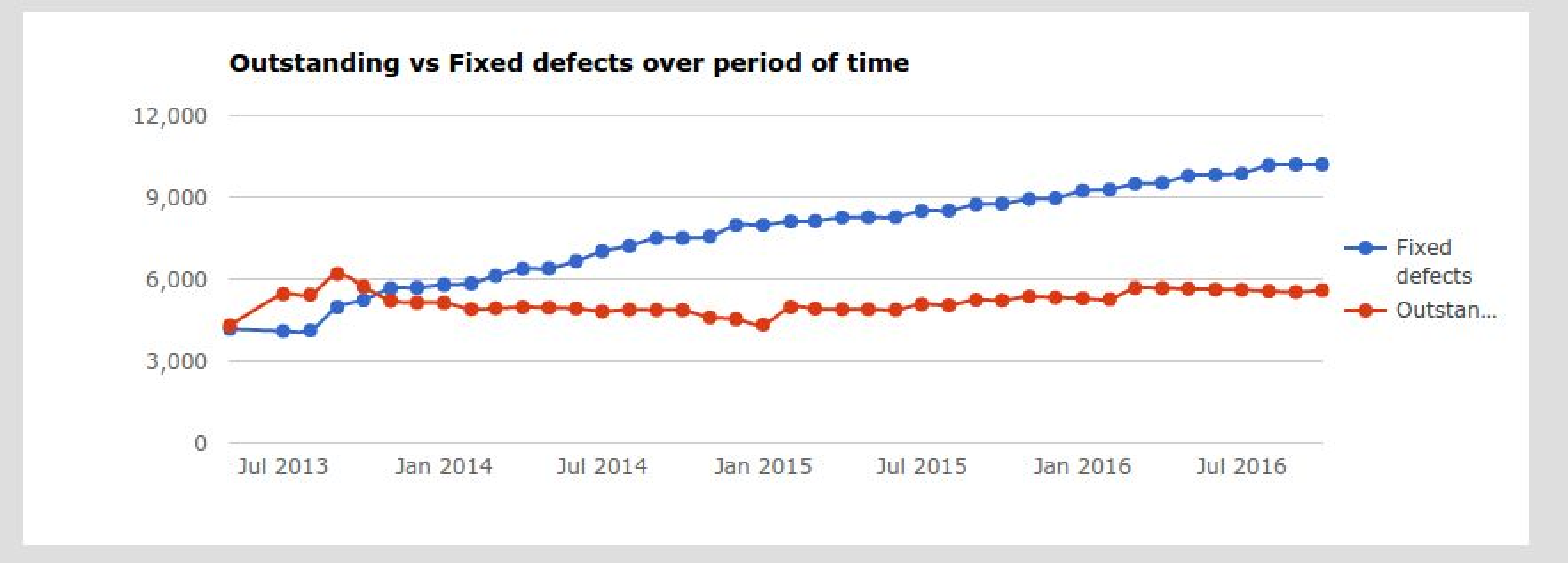
Your Testimonial about Coverity

Let us know how Coverity Scan has helped improve your projects!

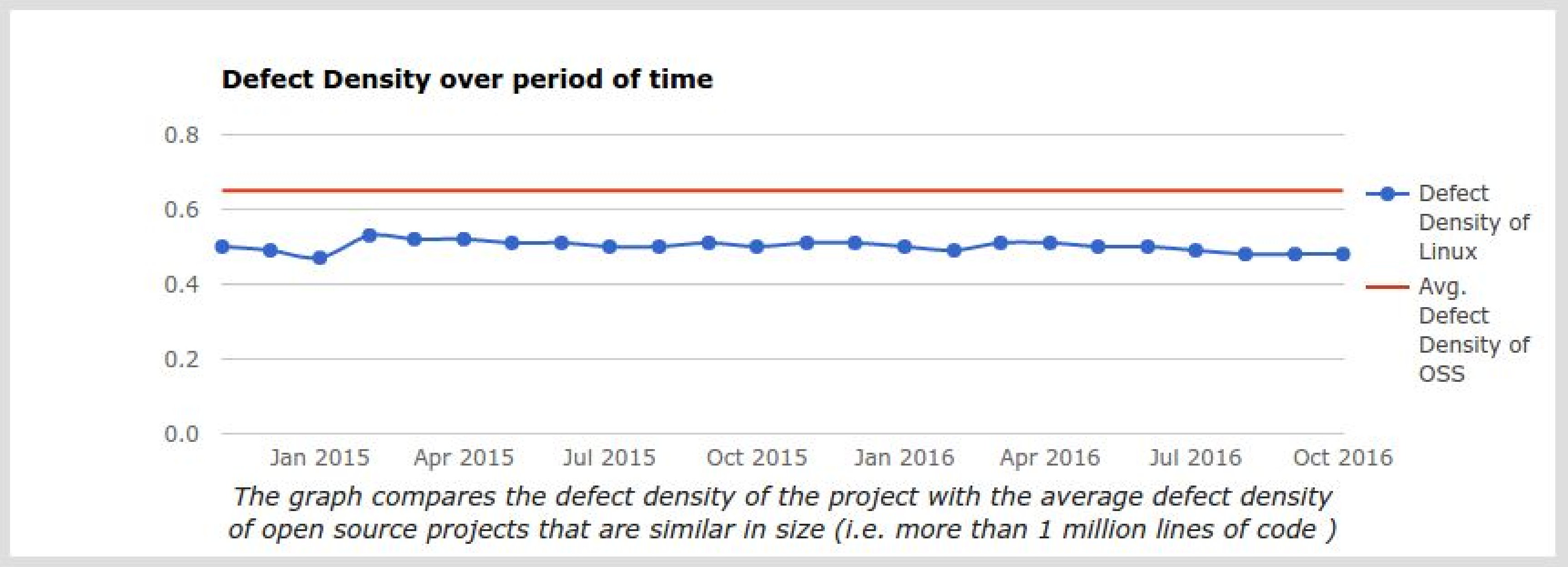
Coverity: Top Defects per Component



Coverity: Outstanding Defects



Coverity: Defect Density



The graph compares the defect density of the project with the average defect density of open source projects that are similar in size (i.e. more than 1 million lines of code)

Summary

Good

- Test coverage has improved significantly over the last 2-3 years
- Test coverage still continuously improving
- The number of kernel bugs (per LOC) follows a downward trend
- People start paying attention to kernel bug reports

Not so Good

- Kernel stability still perceived as insufficient
  - Especially for stable releases
  - Need to further improve test coverage and quality of test reports
- Total number of open defects increases over time
  - Follows kernel code size increase
  - Need to analyze and fix outstanding bugs
- Sometimes it takes a long time for known bugs to get fixed
- No clear guideline how to handle false positives (especially from gcc)
- Not enough people actively engaged in ‘generic’ bug analysis and fixing

Next Steps

- Spread the word
  - Available test suites
  - Testbeds
  - Test coverage
- Improve test coverage
  - Especially but not only for stable releases
- Figure out how to better handle known positives
- Actually *fix* known bugs
  - Bug reports from autobuilders / autobooters
  - Bug reports from static analyzers

Next steps - continued

- Identify and track available test suites
- Improve test coverage
    - More functional tests (both in qemu and on real hardware)
    - Automatic bisect
    - Module tests
        - Implement *and run*
    - More testing on real hardware
- Improve test feedback
    - Automatic reports
    - Unified reporting
    - UI to pull test results

Thank You