Static Analysis of Your OSS Project with Coverity

LinuxCon EU 2015

Stefan Schmidt
Samsung Open Source Group
stefan@osg.samsung.com

Agenda

- Introduction
- Survey of Available Analysers
- Coverity Scan Service
- Hooking it Up in Your Project
- Fine Tuning
- Work Flows & Examples
- Summary

Introduction

Static Analysers

- What is Static Analysis?
  - Analysis of the soure code without execution
  - Usage of algorithms and techniques to find bugs in source code
- What is it not?
  - A formal verification of your code
  - A proof that your code is bug free
- Why is it useful for us?
  - Allows to find many types of defects early in the development process
  - Resource leaks, NULL pointer dereferences, memory corruptions, buffer overflows, etc.
  - Supplements things like unit testing, runtime testing, Valgrind, etc.

Survey of Available Analysers

Static Analysers

- Sparse
- Clang Static Analyzer
- CodeChecker based on Clang Static Analyzer
- Klocwork (proprietary)
- Coverity (proprietary, free as in beer service for OSS projects)
- A list with more analysers can be found at [1]
- My personal experience started with
  - Clang Static Analyzer
  - Klocwork used internally (not allowed to share results)
  - finally settled for Coverity Scan service

Sparse

- Started 2003 by Linus Torvalds
- Semantic parser with a static analysis backend
- Well integrated into the Kernel build system (make C=1/C=2)
- To integrate it with your project using the build wrapper might be enough:

```
make CC=cgcc
```

Clang Static Analyzer

- Command line tool scan-build as build wrapper
- Generates a report as static HTML files
- The analyser itself is implemented as C++ library
- Also used from within XCode
- Scan build had many false positives for us and needs more manual tuning (e.g. leak detected when added to a list or array)
- Turned out to be to noisy without further work for us

CodeChecker

- Recently (June 2015) published by Ericsson
- Based on Clang Static Analyzer library
- Adds database for defect tracking
- Adds interactive web UI for defect handling
- Incremental reporting against baseline
- Added new checkers to Clang itself as well
- Very interesting but sadly no time to test, yet

Feature Comparison

Analyser	OSS	Defect database	Web UI	False positive ratio
Sparse	✓	✗	✗	To be tested
Clang Static Analyzer	✓	✗	✓ static html output	Noisy
CodeChecker	✓	✓	✓	To be tested
Coverity	✗ free as in beer service	✓	✓	Good
Klocwork	✗	✓	✓	Good

Coverity Scan Service

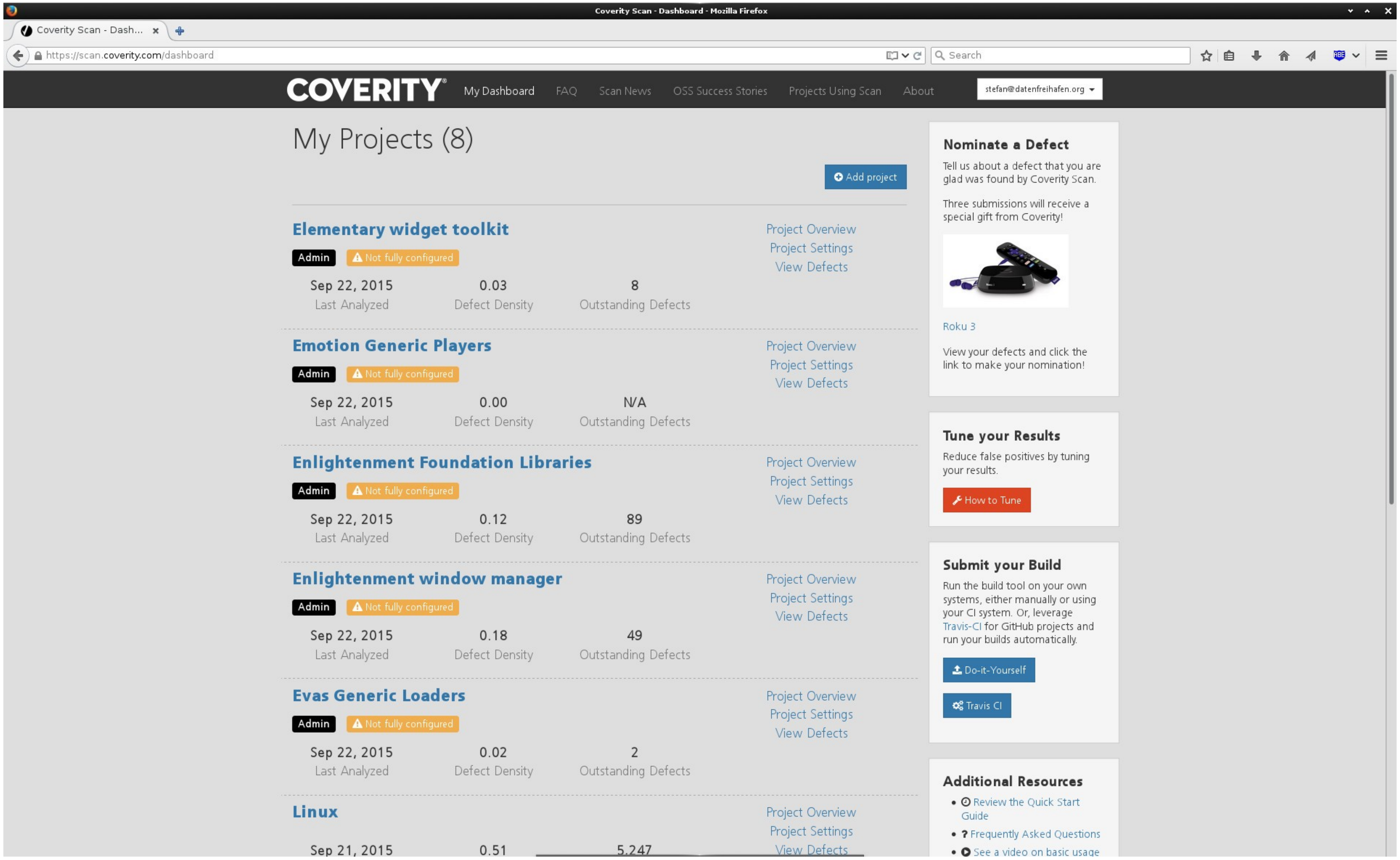
Coverity Scan Service Overview

- Started 2006 with 50 projects and now runs for 5700
- Many big projects already make use of it: Linux, Firefox, LibreOffice, FreeBSD, ...
- Scans projects written in C, C++, Java, C# and JavaScript
- Defect density is defined as defects per 1000 lines of code (1 per 1000 as industry standard)

Coverity Scan Service Parts

1) Build wrapper cov-build to gather data on your system and package it into a tgz file
2) Upload the tgz on the website or via curl to web API to trigger analysis
3) Receive a mail once the analysis is completed
4) Web UI for dashboard and to triage defect reports

Coverity Scan Service Dashboard



My Projects (8)

Add project

Elementary widget toolkit
Admin | Not fully configured
Sep 22, 2015 Last Analyzed | 0.03 Defect Density | 8 Outstanding Defects
Project Overview | Project Settings | View Defects

Emotion Generic Players
Admin | Not fully configured
Sep 22, 2015 Last Analyzed | 0.00 Defect Density | N/A Outstanding Defects
Project Overview | Project Settings | View Defects

Enlightenment Foundation Libraries
Admin | Not fully configured
Sep 22, 2015 Last Analyzed | 0.12 Defect Density | 89 Outstanding Defects
Project Overview | Project Settings | View Defects

Enlightenment window manager
Admin | Not fully configured
Sep 22, 2015 Last Analyzed | 0.18 Defect Density | 49 Outstanding Defects
Project Overview | Project Settings | View Defects

Evas Generic Loaders
Admin | Not fully configured
Sep 22, 2015 Last Analyzed | 0.02 Defect Density | 2 Outstanding Defects
Project Overview | Project Settings | View Defects

Linux
Sep 21, 2015 | 0.51 | 5,247
Project Overview | Project Settings | View Defects

Nominate a Defect

Tell us about a defect that you are glad was found by Coverity Scan.

Three submissions will receive a special gift from Coverity!

Roku 3

View your defects and click the link to make your nomination!

Tune your Results

Reduce false positives by tuning your results.

How to Tune

Submit your Build

Run the build tool on your own systems, either manually or using your CI system. Or, leverage Travis-CI for GitHub projects and run your builds automatically.

Do-it-Yourself

Travis CI

Additional Resources

- Review the Quick Start Guide
- Frequently Asked Questions
- See a video on basic usage

Join a Project

- The simplest way to participate is when the project already uses Coverity Scan
- A good chance as over 5700 projects are registered already
- A searchable list with participating projects can be found at [2]
- Request access, which the project admin might need to approve (depends on project settings)

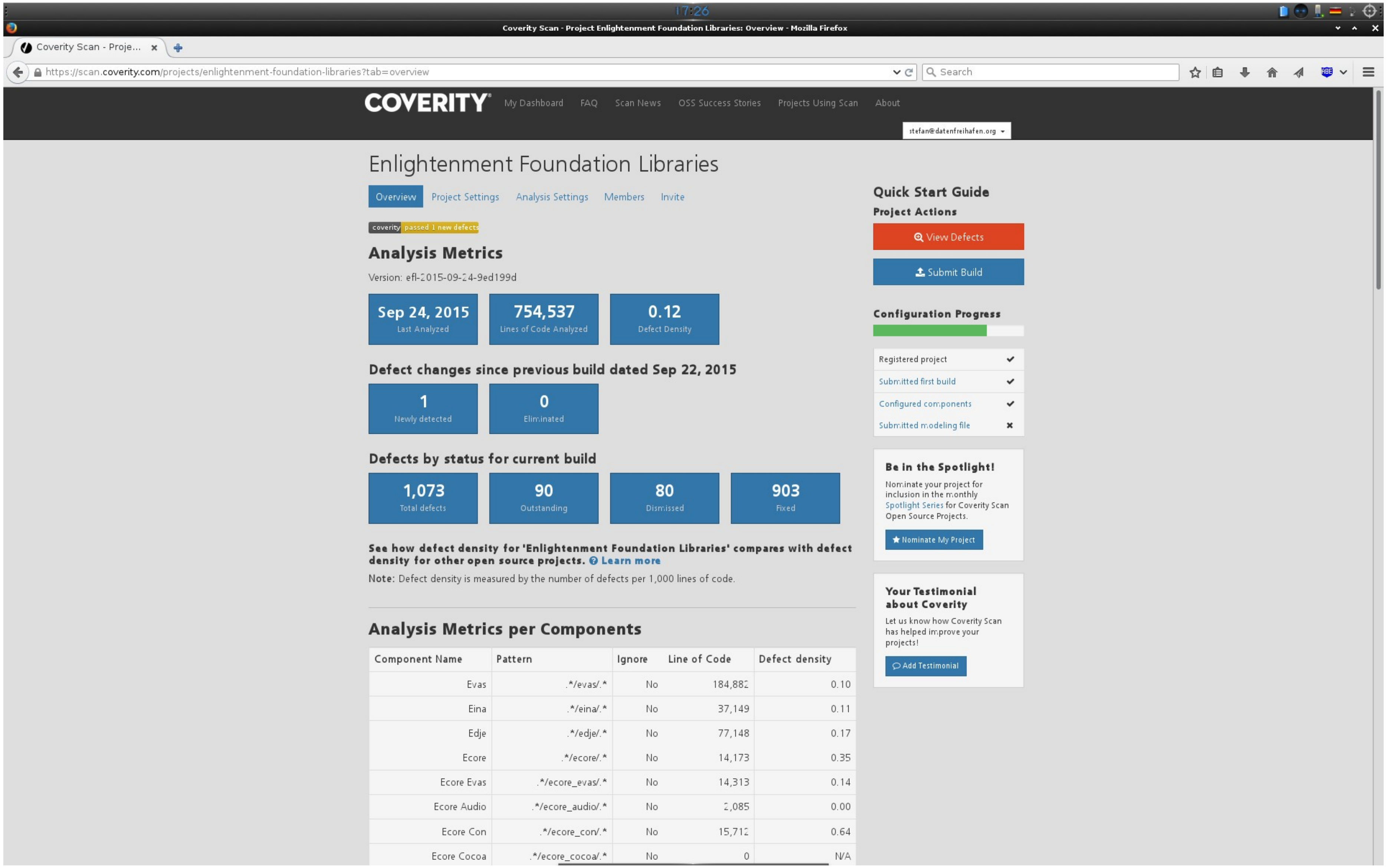
Register a New Project

- If your project is not yet using Coverity Scan you need to register it as a new project at [3]
- Registering is easy (only needs project URL's and license selection)
- It might take a few days until a newly registered project is ready to be analysed
- Once the project has been approved you can submit builds to it

Scan Service Improvements

- Over my 2 years usage of Coverity Scan there have been several improvements hardware and software wise
- Hardware upgrades which results in faster analysis results without long queues
- Improved scanners and heuristics (server side as well as in new cov-build releases) for less false positives
- Graphs in your project view
- Metrics based on defined components
- CWE Top 25 defects

Scan Service Project Page


17:26
Coverity Scan - Project Enlightenment Foundation Libraries: Overview - Mozilla Firefox
Coverity Scan - Proje...
https://scan.coverity.com/projects/enlightenment-foundation-libraries?tab=overview
Search
COVERITY
My Dashboard
FAQ
Scan News
OSS Success Stories
Projects Using Scan
About
stefan@datenfreihafen.org

Enlightenment Foundation Libraries

Overview | Project Settings | Analysis Settings | Members | Invite

coverity passed 1 new defects

Analysis Metrics

Version: efl-2015-09-24-9ed199d

Sep 24, 2015	754,537	0.12
Last Analyzed	Lines of Code Analyzed	Defect Density

Defect changes since previous build dated Sep 22, 2015

1	0
Newly detected	Eliminated

Defects by status for current build

1,073	90	80	903
Total defects	Outstanding	Dismissed	Fixed

See how defect density for 'Enlightenment Foundation Libraries' compares with defect density for other open source projects. Learn more

Note: Defect density is measured by the number of defects per 1,000 lines of code.

Analysis Metrics per Components

Component Name	Pattern	Ignore	Line of Code	Defect density
Evas	.*/evas/.*	No	184,882	0.10
Eina	.*/eina/.*	No	37,149	0.11
Edje	.*/edje/.*	No	77,148	0.17
Ecore	.*/ecore/.*	No	14,173	0.35
Ecore Evas	.*/ecore_evas/.*	No	14,313	0.14
Ecore Audio	.*/ecore_audio/.*	No	2,085	0.00
Ecore Con	.*/ecore_con/.*	No	15,712	0.64
Ecore Cocoa	.*/ecore_cocoa/.*	No	0	N/A

Quick Start Guide

Project Actions

View Defects

Submit Build

Configuration Progress

Registered project	✓
Submitted first build	✓
Configured components	✓
Submitted modeling file	✗

Be in the Spotlight!

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

Nominate My Project

Your Testimonial about Coverity

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

Add Testimonial

Hooking it Up in Your Project

Gather Build Data

- To gather the data needed by the analyser Coverity provides a build wrapper
- Cov-build needs to be run with your normal build tools as parameter
- If you project uses make it should be as easy as:

```
cov-build --dir cov-int make
```

- It is updated twice a year and recommended to keep your version up to date [4]

Manually Submit Builds

- You can submit builds manually through the web interface
- Just upload it from the Submit Build form from your project overview page
- This make sense for your first builds or if you want to test something
- In general the process should better be automated

Submit Builds with Travis CI

- Travis CI build system integrated with GitHub
- Very useful if you use GitHub and/or Travis
- You need to setup your project in Coverity Scan as GitHub project to have the Travis option available
- Operates on a per-branch basis (default name coverity_scan)
- Once you push your code to this branch on GitHub Travis will trigger the Coverity Scan run on it
- A full guideline with .travis.yml template can be found at [5]

Submit Builds from Jenkins

- There exists a Coverity Plugin for Jenkins [6]
- At the time I tried it, I was not able to use the free Scan Service as Integrity Manager instance
- Seems it was only capable of integrating with a commercial license on your setup

Submit Builds from Jenkins

- Simply used cov-build and curl to generate and upload the data to Coverity Scan

```
FILENAME=efl-$(date -I)-$(git rev-parse --short HEAD)

rm -rf cov-int

./autogen.sh --prefix="${EFL_DESTDIR}" ${config_opts}

cov-build --dir cov-int make -j${PARALLEL_MAKE}

tar czvf $FILENAME.tgz cov-int

curl --form token=XXX --form email=stefan@datenfreihafen.org --form file=@$FILENAME.tgz --for
m version=$FILENAME --form description=$FILENAME https://scan.coverity.com/builds?project=Enli
ghtenment+Foundation+Libraries

make -j${PARALLEL_MAKE} distclean
```

Fine Tuning

Fine Tuning on the Server

- Create project components
  - Simple regex patterns to sort files into categories
  - Useful for large code bases
  - Useful for projects with many maintainers
- You can create a modeling file to adjust
  - Helps to tune down the false positive rate
  - Upload a file to annotate functions without implementation for things like abort, free or alloc
  - I had no need for it until now

Fine Tuning in the Code

- Annotations in code
  - Better use the modeling file (keeps code clean)
  - +kill (always aborts), +alloc (allocates memory), +free (frees argument)

```
/* coverity[+free : arg-0] */
void local_free(void *to_be_freed) {
...
}
```

- Mention the unique CID's in commit messages for credit and backreferencing

Work Flows & Examples

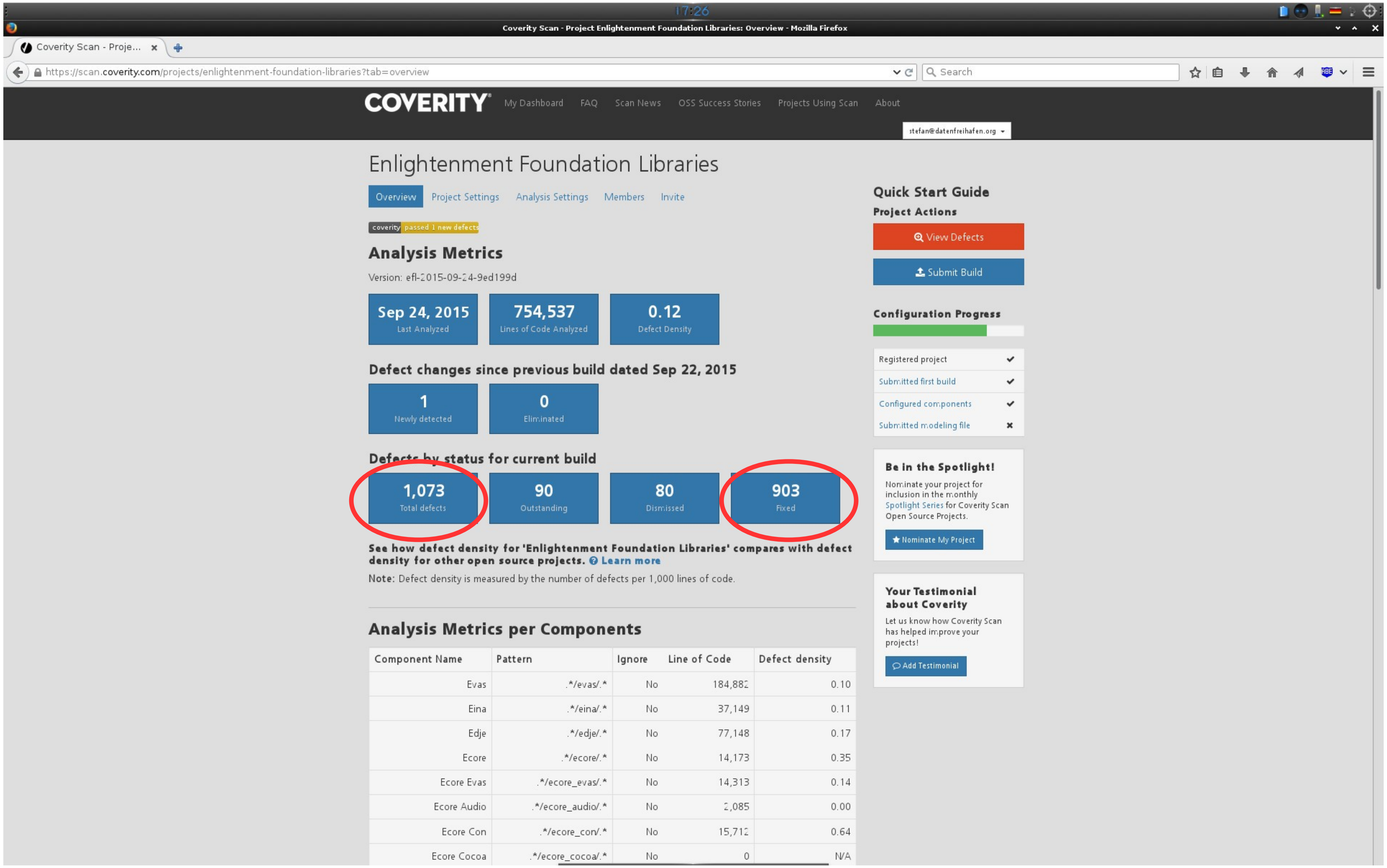
Work Flow - EFL

- Started to use it in July 2013 with the Enlightenment Foundation Libraries
- 7 projects from 32k to 750k lines of code
- 3 of them reached a 0 defect rate the rest ranges from 0.02 to 0.18
- Submitted every night from our Jenkins CI setup (one project is to big > 500k LOC and thus can only run 4 times a week)
- Mail with scan results is send to a mailing list
- Normally new reports get fixed quickly as they are in areas which are actively being worked on

Work Flow - EFL

- During the stabilization phase of our development cycle I go through the list and dispatch defects with high impact
- Would love to run new patch submissions through the scan during review
  - To much load towards the scan service
  - Incremental checks would be interesting as well

Work Flow - EFL Example



17:26
Coverity Scan - Project Enlightenment Foundation Libraries: Overview - Mozilla Firefox
Coverity Scan - Proje...
https://scan.coverity.com/projects/enlightenment-foundation-libraries?tab=overview
Search
COVERITY
My Dashboard
FAQ
Scan News
OSS Success Stories
Projects Using Scan
About
stefan@datenfreihafen.org

Enlightenment Foundation Libraries

Overview
Project Settings
Analysis Settings
Members
Invite
coverity passed 1 new defects

Analysis Metrics

Version: efl-2015-09-24-9ed199d

Sep 24, 2015	754,537	0.12
Last Analyzed	Lines of Code Analyzed	Defect Density

Defect changes since previous build dated Sep 22, 2015

1	0
Newly detected	Eliminated

Defects by status for current build

1,073	90	80	903
Total defects	Outstanding	Dismissed	Fixed

See how defect density for 'Enlightenment Foundation Libraries' compares with defect density for other open source projects. Learn more

Note: Defect density is measured by the number of defects per 1,000 lines of code.

Analysis Metrics per Components

Component Name	Pattern	Ignore	Line of Code	Defect density
Evas	.*/evas/.*	No	184,882	0.10
Eina	.*/eina/.*	No	37,149	0.11
Edje	.*/edje/.*	No	77,148	0.17
Ecore	.*/ecore/.*	No	14,173	0.35
Ecore Evas	.*/ecore_evas/.*	No	14,313	0.14
Ecore Audio	.*/ecore_audio/.*	No	2,085	0.00
Ecore Con	.*/ecore_con/.*	No	15,712	0.64
Ecore Cocoa	.*/ecore_cocoa/.*	No	0	N/A

Quick Start Guide

Project Actions

Configuration Progress

Registered project	✓
Submitted first build	✓
Configured components	✓
Submitted modeling file	✗

Be in the Spotlight!

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

Your Testimonial about Coverity

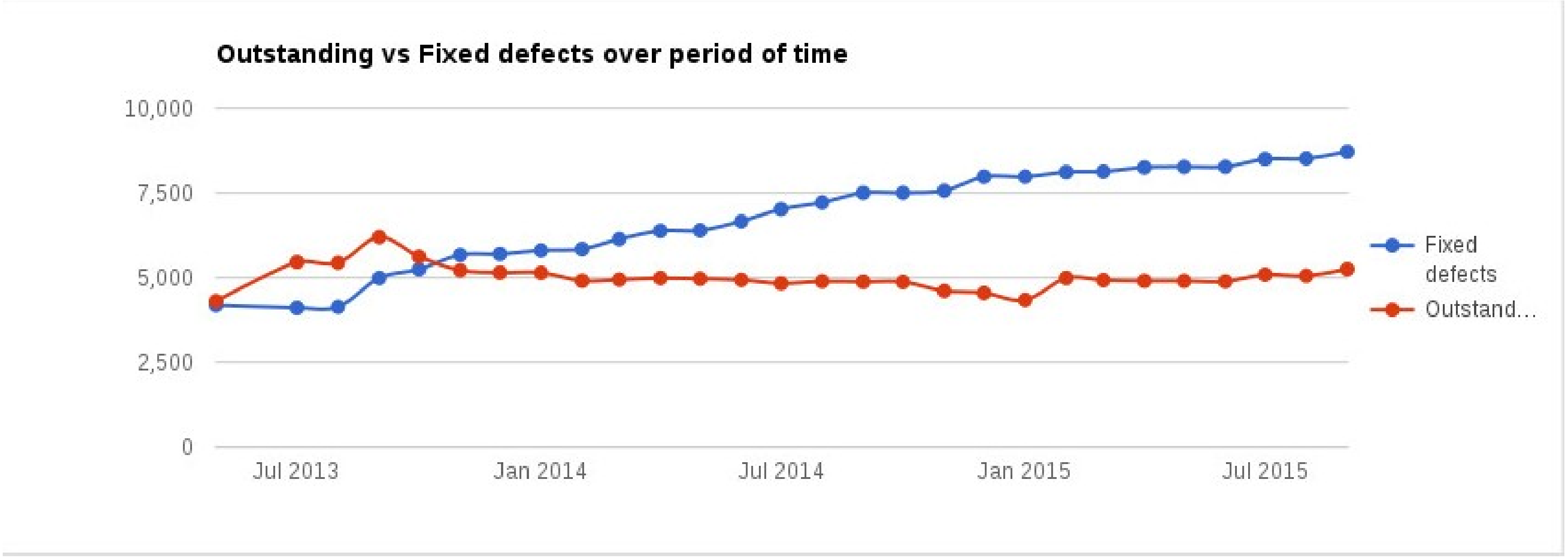
Let us know how Coverity Scan has helped improve your projects!

Work Flow - Linux

- Huge code base with ~10M lines of code (after C preprocessor)
- Build submitted once a week by Dave Jones
- Many maintainers and developers accessing it directly and looking at their components
- Fixes come through the normal dev channels

Work Flow - Linux

- Defect level is staying around 5000 for a long time now
- Hard to fix obscure areas without domain knowledge or hardware drivers without hardware
- Much old code



Work Flow - Alternatives

- Run every commit through it
  - Most likely overkill and will not really work well with the free Scan Service
- Dedicated git branches to be checked
  - Only works with git
  - The way the Travis CI plugin works
  - Maybe interesting for testing review branches

Striving for 0

- Striving for defect rate of 0
- Gamification
- We have reached this in three of the smaller projects
- Harder to reach in large and old code bases
- Once reached, higher motivation to look at new defects to maintain the 0 defect rate
- This can obviously only cover problems found by Coverity Scan. You surely have more. :-)

Defect Areas

- In my experience the majority of defects are in seldomly used code paths or new code
- Which explains why they are still there
- An example would be resource leaks on error paths and during shutdown
- On every 10 or 20 of those defects though there comes one which makes you really wonder how it could be in your code at all :-)
- Some stories at [7]

Examples

- Classic resource leaks
  - Not seen to often if you regularly run your code under Valgrind
- Buffer overruns and memory corruptions
  - Good to find those early-on instead of having to go through a lengthy debug session
- Copy and paste defects which result in logic flaws

Summary

Summary

- Using a static analyser is a good addition to your QA toolset
- The setup and usage is easy enough and gives you a quick and direct benefit
- Finds defects early in the process instead of during deployment
- Various alternatives to Coverity Scan if they fit you better
- Recommended to run regularly

References

- [1]: https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
- Sparse: https://sparse.wiki.kernel.org/index.php/Main_Page
- Clang Static Analyzer: http://clang-analyzer.llvm.org
- CodeChecker: https://github.com/Ericsson/codechecker
- Coverity Scan: https://scan.coverity.com
- [2]: https://scan.coverity.com/projects
- [3]: https://scan.coverity.com/projects/new
- [4]: https://scan.coverity.com/download?tab=cxx
- [5]: https://scan.coverity.com/travis_ci
- [6]: https://wiki.jenkins-ci.org/display/JENKINS/Coverity+Plugin
- [7]: https://scan.coverity.com/o/oss_success_stories

Thank you.