coverity®

Coverity Scan:

2013 Open Source Report

"Coverity Scan helped us identify multiple potential buffer overrun vulnerabilities, due to integer overflow in size calculations."
- PostgreSQL -

"The open source tools are good, and improving, but Coverity currently provides a superior experience."
- Vincent Sanders (Netsurf Browser contributor) -

"If you contribute to an open source project, you should be using Coverity Scan. It will likely find bugs that can certainly have security implications in your code."
- fwknop -

"The reports from Coverity are a valuable contribution to - among others - the LibreOffice development process."
- LibreOffice -

"Coverity remains the single most useful tool I've used."
- Ward Fisher (NetCDF contributor) -

Coverity Scan: A Brief Introduction

The Coverity Scan™ service began as a public-private sector research project, focused on open source software quality and security. Initiated in 2006 with the U.S. Department of Homeland Security, Coverity now manages the project, providing our development testing technology as a free service to the open source community to help them build quality and security into their software development process. The Coverity Scan service enables open source developers to scan–or test–their Java, C and C++ code as it is written, flag critical quality and security defects that are difficult (if not impossible) to identify with other methods and manual reviews, and provide developers with actionable information to help them to quickly and efficiently fix the identified defects.

In 2013, the Coverity Scan service experienced explosive growth and now has more than 1,500 projects.

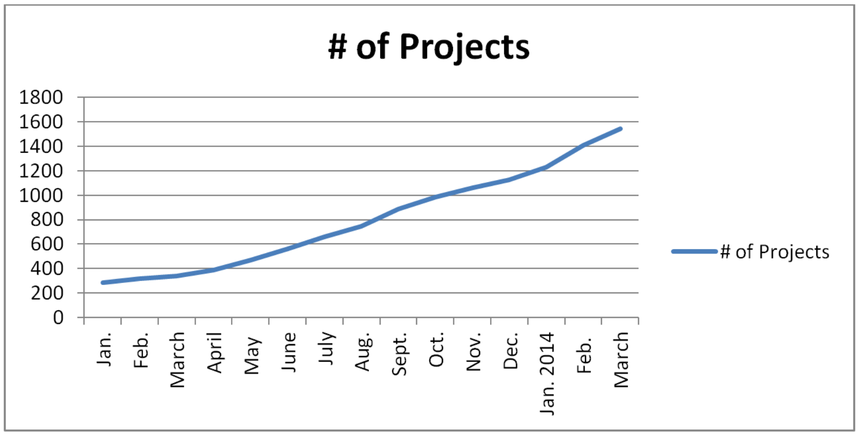

Table 1: Total number of projects in Coverity Scan service through March 2014

This growth reaffirms the power and importance of development testing, and static analysis as a foundational technology, for open source projects to assure the quality and security of their code.

In May of 2013, we expanded the service to include support for Java. Since that time, more than 100 open source projects have joined the service including popular Big Data projects such as Apache Hadoop, Apache HBase and Apache Cassandra, and other widely adopted Java projects such as Apache CloudStack, as well as Hudson Server, Eclipse Code Recommender and many others. In less than one year, our new Java members fixed thousands of defects. Coverity also joined the Eclipse Foundation and created a Coverity Scan Hudson plugin that integrates with projects hosted by the Eclipse Foundation, making it easier for these projects to build development testing into their development workflow.

Open Source Software is Eating the World

If software is eating the world, as posited by Marc Andreessen, then open source software is leading the charge. According to a recent study by Black Duck Software and North Bridge Venture Partners, the adoption and support of open source software in the enterprise has never been higher. Eight out of ten people surveyed are choosing open source software based on quality. Open source software has become a key component in the enterprise software supply chain and is critical to the success of many organizations. As evidence of this, in September of 2013, IBM announced plans to invest USD $1 billion in Linux and open source technologies for their Power Systems servers.

Coverity Scan: Open to a New Audience

The Coverity Scan service has been impacted by this phenomenon as well. In the past year, we've seen an exponential increase in the number of people who have asked to join particular projects simply to monitor the defects being found and fixed. In many cases, these people work for large enterprises that utilize open source software within their commercial solutions, and thus have a deep interest in understanding and monitoring the quality of these projects. Up until now, Project Administrators were required to approve any new member who wanted to contribute or simply monitor a project. Based on the overwhelming number of requests received, as well as the positive feedback from Scan Project Administrators, we are now enabling individuals to become Project Observers. This means they can track the state of their favorite open source projects in the Scan service and view high-level data including the count of outstanding defects, fixed defects and defect density.

We believe allowing individuals to view this type of information about specific projects will help them to make informed decisions, based on some key measures of code quality, about which projects they should include in their software supply chain. Black Duck Software, a Coverity Certified Partner, provides complementary capabilities to help assess the quality and stability of open source projects.

A Healthy Community Matters

By Black Duck Software

Having an active, vibrant community behind an open source software (OSS) project is a critical factor in determining that project's overall health. While assessing defect density data provides an important perspective, assessing its community health helps further determine if issues will be resolved quickly and if future feature investment is likely to occur.

Healthy projects have multiple active contributors who make frequent commits, and most of the highest quality projects have a diverse contributor base, offering multiple perspectives and use cases. Tracking and assessing these community characteristics requires insight into a project's commit and contributor activity and history. You can find this data on sites like Ohloh.net and GitHub.

Newer, growing projects often exhibit different community characteristics versus older, more mature projects. For new projects, community health can often be assessed by the number of net-new contributors and the frequency and amount of code contributions made. As projects mature, the focus often shifts to larger numbers of people contributing to bug fixes, documentation updates, integration elements and other adjacent contributions.

Whether project is big or small, new or old, its commit details, activity levels and contributor details provide valuable insight into a project's community health. Our mission with the Coverity Scan service is to help further the rapid adoption of open source software by providing open source projects with the ability to build code quality and security into the development process in the same way as commercial projects. Based on the millions of lines of open source software we have scanned over the past eight years, we have become an authority on the state of open source software quality. Our aim is to provide an objective industry standard that developers can use to evaluate open source software quality and ultimately increase the adoption of open source code.

Enhancements

In addition to introducing this new level of visibility, we made numerous enhancements to our Scan service to improve ease of use and enable more developers to be able to get started and begin viewing defects with just a few clicks of a button. We recently developed integrations with GitHub and Travis CI so that developers can leverage our combined cloud-based platforms to write their code, build their applications and find and fix high-impact defects before releasing their software. More than 40 percent of the projects using the Scan service currently have repositories on GitHub and this new, single sign-on capability will make it even easier for their contributors to view and address high-impact defects in their code.

We also upgraded our static analysis engine to take advantage of the new and tuned algorithms for Java, C and C++, and made improvements to our build process to make it easier for administrators to run the analysis on their own.

Project and Developer Adoption

The total number of developers using the Coverity Scan service has now grown to almost 3,500 users as of March 2014. However, this still doesn't capture the full impact of the Scan service as it only represents the number of people who access our system to capture defect information. It does not capture all of the individual developers who are working to fix the defects we have identified. Many projects have assigned quality specific roles and it is this group of individuals who typically access our service. They review all of the new defects and then publish the results to a much broader set of developers, so they can in turn collaborate on resolving the identified defects. Some projects, such as Python, have created an email alias where they share all newly detected defects, others publish the results on their project page under a separate Coverity Scan service section, while many others simply forward the list of defects to an internal email list.

Saving Time and Resources
One very large project in the Scan service, unfortunately, did not have a policy to review all new defects. If it had, in February of 2013, it would have been able to address a high-impact defect that could have led to a denial of service or allowed the program to execute arbitrary via a crafted ImageText request that triggers memory-allocation failure. Instead, the project discovered and addressed the issue in October of 2013, which surely involved far more resources and left the project users vulnerable to the defect for eight months.

We've seen an important trend with our open source projects that has been mirrored by our commercial customers. Coverity has long recommended a "no new defects" policy as a best practice for driving adoption of development testing. For many, the vast quantities of defects that are discovered the first time the Coverity analysis is run can be overwhelming. When faced with thousands of defects, it's difficult to know where to start. By putting in place a "no new defects policy," it is far easier to address new defects because the source code is still fresh in the mind of the developer and therefore will take less time to fix. In addition to ensuring no new defects are introduced, teams then also implement a plan for working on the backlog of issues, starting with the high-impact and most critical defects.

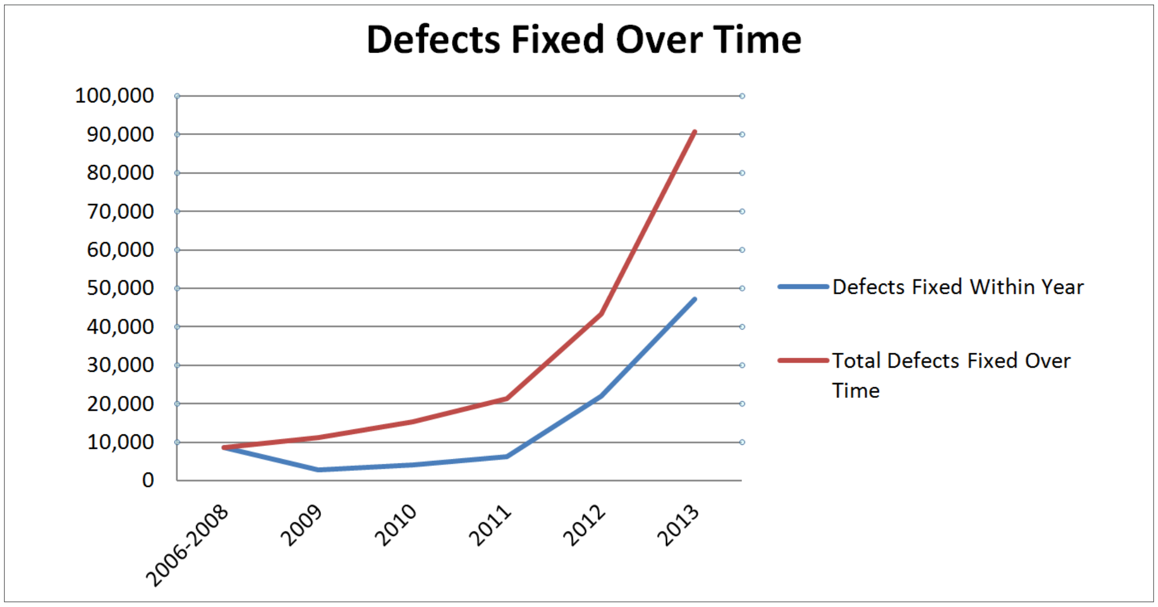

Table 2: All Defects Fixed From 2008-2013

Almost 50,000 defects were fixed in 2013 alone – more than the amount of defects that had been fixed in the history of the service.

The State of Open Source Software Quality: C/C++

In 2013, we analyzed more than 250 million lines of code from more than 740 of our most active C and C++ projects, which represented a mix of long-time and newer members of the service. The projects range in size from fewer than 5,000 lines of code to more than 16 million lines of code (for NetBSD). The next three largest projects in the service are FreeBSD, LibreOffice and Linux, with 12, 9 and 8.5 million lines of code respectively. Due to the magnitude of the increase in the number of projects analyzed for this report compared to the 118 that were included last year, we saw some shifts in the distribution of project sizes. We continued to see large numbers of mid- to smaller-sized projects join the program and adopt static analysis as a software development best practice.

TABLE 3: 2013 PROJECT DISTRIBUTION BY CODEBASE SIZE – C/C++

Size of Codebase (Lines of Code)	Number of Projects 2013	% of Total Projects 2013	% of Total Projects 2012
Less than 100,000	358	48%	38%
100,000-499,999	299	40%	44%
500,000-1 million	42	6%	7%
More than 1 million	42	6%	11%
Total	741		

A Note on False Positive Rates

The leading barrier to adoption of static analysis technology has historically been the accuracy of analysis. Developers will not use a solution if it produces large volumes of false positive results–there is simply too high a signal to noise ratio. Therefore, the accuracy of our analysis is of critical importance to us. With every release, we tune our analysis results to further improve the breadth and depth of our defect detection. We use the more than 250 million lines of open source code and 5 billion lines of proprietary code we've analyzed to help us continually improve our analysis algorithms. Our false positive rates have been declining over the years. In the 2008 report, we noted the average false positive rate across all open source projects was 13.2%; in the 2009 report, that number dropped to 9.8% and in 2012, the rate dropped again to 9.7%. In 2013, we saw the false positive rate take a slight uptick to 10.2%. We believe this increase is a result of the large number of new analysis algorithms that will be tuned over time with large volumes of use, as well as the large influx of new projects which joined the Scan service. In the early stages of participation in the service, many projects haven't yet created the models which enable our analysis engine to better understand the specific idioms of individual projects. Despite the slight increase, our false positive rate continues to be far below that of most commercial and open source static analysis solutions available today.

TABLE 4: AVERAGE DEFECT DENSITY ACROSS ALL ACTIVE SCAN C/C++ PROJECTS 2008-2013

Coverity Scan Report Year	Average Defect Density C/C++
2008	.30
2009	.25
2010	.81
2011	.45
2012	.69
2013	.59

We witnessed defect density for open source projects in C and C++ drop this year, despite the influx of new projects. We have historically seen that codebases with a defect density of 1.0, or 1 defect per every 1,000 lines of code, are considered good quality software. Last year, we stated that based on our historical data, open source software (for projects which have adopted development testing via the Coverity Scan service) not only has better than average quality as compared to the industry average, but in fact continues to raise the bar on what is considered good quality software for the entire industry. That trend continues as the defect density of the open source projects participating in the Scan service dropped to .59 in 2013.

TABLE 5: 2013 DEFECT DENSITY BY PROJECT SIZE – C/C++

Size of Codebase (Lines of Code)	Defect Density 2013	Defect Density 2012
Less than 100,000	.35	.40
100,000-499,999	.50	.60
500,000-1 million	.70	.44
More than 1 million	.65	.75
Average Across Projects	.59	.69

Commentary on Defect Density:
In 2013, the overall defect density for C and C++ projects was lower than in 2012 for all but one level. We had five times more projects with 500,000-1 million lines of code. With new projects, it is not uncommon to see higher defect density rates than in projects that have been using static analysis through the Scan service for several years, and have matured their usage such that static analysis is used regularly as part of their standard process.

Defect density in this report is measured by the number of defects per 1,000 lines of code identified by the Coverity Development Testing Platform. It does not include defects found through other testing methods or post-deployment use. Defect density is computed using only defects in the "high-impact" or "medium-impact" categories for C/C++ projects and is a measure of confirmed and potential defects that are left in the codebase at the time of this report.

Open Source Code Quality Surpasses Proprietary Code Quality In C/C++ Projects

In 2013, for the first time, we saw open source quality for the projects in the Scan service surpass that of proprietary projects at all code base sizes. The 2012 Coverity Scan Report looked at a sample analysis of more than 250 proprietary code bases totaling more than 380 million lines of code, with an average codebase of nearly 1.5 million lines of code, and we found that open source code had lower defect density levels up to 1 million lines of code. For the 2013 report, we analyzed approximately 500 million lines of code across almost 500 proprietary C/C++ projects.

TABLE 6: 2013 COMPARISON OF OPEN SOURCE AND PROPRIETARY C/C++ CODE

Size of Codebase (Lines of Code)	Open Source Code	Proprietary Code
Lines of Code	252,010,313	684,318,640
Number of Projects	741	493
Average Project Size (lines of code)	340,094	1,388,070
Defects Outstanding as of 12/31/13	149,597	492,578
Defects Fixed in 2013	44,641	783,799
Defect Density	.59	.72

In 2013, the defect density rate of open source code was lower than that of proprietary code. According to recent reports sponsored by Black Duck Software and North Bridge Venture Partners, quality concerns are no longer a barrier to open source adoption in the enterprise. In fact, the quality of the open source code for Coverity Scan participants can be higher than the proprietary code included in an enterprise product.

TABLE 7: 2013 DEFECT DENSITY BY PROJECT SIZE OPEN SOURCE VS. PROPRIETARY C/C++ CODE

Size of Codebase (Lines of Code)	Open Source	Proprietary Code
Less than 100,000	.35	.38
100,000-499,999	.50	.81
500,000-1 million	.70	.84
More than 1 million	.65	.71
Average across projects	.59	.72

In 2013, we saw the quality of open source code surpass that of proprietary code at every level. We did not see the same cliff in terms of quality for projects with more than one million lines of code. One of the key factors leading to this change was the overall dedication to quality by our largest projects. NetBSD, FreeBSD, LibreOffice and Linux collectively fixed more than 11,000 defects in their code in 2013.

Most Commonly Fixed Defects by Type

The top defects fixed in C and C++ projects continued to hold steady with resource leaks, null pointers and control flow issues being the three most commonly fixed issues. However the rank order and sheer volume of such issues changed dramatically from 2012. More than three times the number of resource leaks were fixed compared to 2012. Overall, more than double the amount of defects were fixed in C and C++ code in comparison to 2012. For those not familiar with defect types and their potential impact:

- *Resource leaks* often involve failure to release resources when the initial allocation succeeds, but a subsequent, additional resource is not available. This type of defect can impact system reliability and stability, or cause a program crash, due to the behavior of the code that fails when resources cannot be allocated. These malfunctions are often very difficult to reproduce in a non-production environment.

- *Null pointer dereferences* often occur when one code path initializes a pointer before its use, and another code path bypasses the initialization process. These defects are relatively easy to fix when they are found, however, they are often missed in code reviews because they may occur only on certain code paths. If these paths are never exercised during testing, these defects can be the cause of crashes in the user's environment that are not easily replicated and thus not easily remedied.

- *Control flow* issues include defects in which the program contains code that either never executes, such as dead code, or executes under the wrong conditions and could lead to unexpected behavior or security vulnerabilities.

37% of all C and C++ defects fixed in 2013 were classified as high-impact defects, up slightly from 36% in 2012. The top three types of issues fixed in 2013 were consistent with the 2012 Coverity Scan report.

TABLE 8: DEFECTS FIXED BY TYPE AND IMPACT IN C/C++ CODE IN 2013

Category	Quantity 2013	Quantity 2012	Impact
Resource leaks	9,503	2,544	High
Null pointer dereferences	6,573	2,724	Medium
Control flow issues	5,175	3,464	Medium
Error handling issues	4,500	1,432	Medium
Uninitialized members	3,398	918	Medium
Memory - illegal accesses	2,591	1,693	High
Memory - corruptions	2,555	2,264	High
Integer handling issues	2,448	2,512	Medium
Uninitialized variables	1,997	1,374	High
Incorrect expression	1,912	766	Medium
Code maintainability issues	982	476	Low
Security best practices violations	759	254	Low
Insecure data handling	705	751	Medium
API usage errors	619	257	Medium
Program hangs	324	127	Medium
Concurrent data access violations	164	175	Medium
Performance inefficiencies	153	49	Low
Parse warnings	260	188	Low
Class hierarchy inconsistencies	17	4	Medium
Total	**44,641**	**21,972**	

Defects Outstanding

The number of defects outstanding increased dramatically to more than 156,000, up from 21,972 at the end of 2012. This increase is a reflection of the large number of new projects which joined the service in 2013, and the dramatic increase in lines of code analyzed. The top outstanding defects in 2013 were fairly consistent with 2012. The only difference was more uninitialized member defects were left outstanding in 2013 versus 2012. Control flow issues dropped to the third position for highest type of outstanding issue.

TABLE 9: DEFECTS OUTSTANDING BY TYPE AND IMPACT AS OF DEC. 31, 2013 IN C/C++ CODE

Category	Quantity	Impact
Null pointer dereferences	19,574	Medium
Uninitialized members	13,967	Medium
Control flow issues	12,015	Medium
Error handling issues	12,679	Medium
Resource leaks	11,318	High
Integer handling issues	5,329	Medium
Memory - illegal accesses	4,957	High
Memory – corruptions	4,434	High
Uninitialized variables	4,299	High
Security best practices violations	3,750	Low
Insecure data handling	3,611	Medium
Incorrect expression	3,165	Medium
Code maintainability issues	2,818	Low
API usage errors	2,665	Medium
Performance inefficiencies	723	Low
Program hangs	688	Medium
Concurrent data access violations	611	Medium
Parse warnings	454	Low
Class hierarchy inconsistencies	69	Medium
Total	**107,126**	

Linux: Through the Years

Coverity and Linux have had a long standing partnership on code quality which started in the early 2000's when Coverity was still a research project in the Computer Science Laboratory at Stanford University. Since that time, both Coverity and Linux have experienced tremendous growth. One thing that has remained constant for both parties is our commitment to quality.

TABLE 10: LINUX ANALYSIS 2006-2013

Year	Version(s)	Lines of Code Analyzed	New Defects Identified	Defects Fixed
2006	2.6.16	3,451,730	1,264	435
2007	2.6.16	3,458,369	425	217
2008	2.6.27	4,202,209	596	365
2009	2.6.32	4,862,567	527	417
2010	2.6.33, 2.6.34, 2.6.35	5,504,780	3,858	462
2011	2.6.38, 2.6.39, 3.0, 3.1	6,849,378	2,331	1,283
2012	3.2, 3.3, 3.4, 3.5, 3.6., 3.7	7,387,908	5,803	5,170
2013	3.12	8,578,254	3,299	3,346

TABLE 11: LINUX DEFECTS FOUND AND FIXED BY TYPE AND IMPACT IN 2013

Category	Outstanding	Fixed	Impact
Control flow issues	1,071	301	Medium
Integer handling issues	626	816	Medium
Null pointer dereferences	543	291	Medium
Memory - illegal accesses	509	373	High
Error handling issues	449	220	Medium
Incorrect expression	378	92	Medium
Insecure data handling	351	56	Medium
Memory – corruptions	316	762	High

Uninitialized variables	296	207	High
Code maintainability issues	237	57	Low
Resource leaks	158	128	High
Security best practices violations	102	25	Low
Program hangs	21	3	Medium
API usage errors	18	11	Medium
Concurrent data access violations	12	3	Medium
Performance inefficiencies	0	1	Low
Grand Total	**5,092**	**3,346**	**Defect Density: 0.61**

To help ensure highly accurate static analysis results in the Scan service, the Linux team leverages Coverity's modeling capabilities to help the analysis algorithms better understand the patterns and behavior of the Linux code. The analysis automatically builds models based on the source code, but it can't always correctly infer what happens–perhaps there is no source code, like in the case of a dynamic library, or there are external effects that cannot be predicted, such as a remote procedure call.

When Dave Jones, Coverity Scan Administrator for Linux, first became involved in the Coverity Scan service in August of 2013, he established many new, smaller components for Linux to simplify management. Table 12 shows the defect density for the ten largest components. The Linux kernel continues to improve in quality each year. In 2011, it had a defect density of .95, which dropped to .76 in 2012 and is now .61. This change is testament to the team's commitment to quality.

TABLE 12: LINUX DEFECT DENSITY BY COMPONENT 2013

Component	Linux 2013
Drivers	0.58
Drivers-Media	0.47
Drivers-SCSI	0.67
Staging-lustre	0.85
Drivers-Staging	0.77
Networking	0.50
Drivers-USB	0.49

Kernel	0.61
Drivers-Infiniband	0.65
Drivers-USB	0.49
Kernel	0.61
Drivers-Infiniband	0.65

One of the other key changes implemented by Dave was the focus on rapidly eliminating newly detected defects. He believes that newly detected defects are easier to resolve because the code is still fresh in the mind of the developer and legacy defects can often take care of themselves over time, as the code is rewritten and the defect is eliminated. The following chart shows the impact of this focus on new defects:

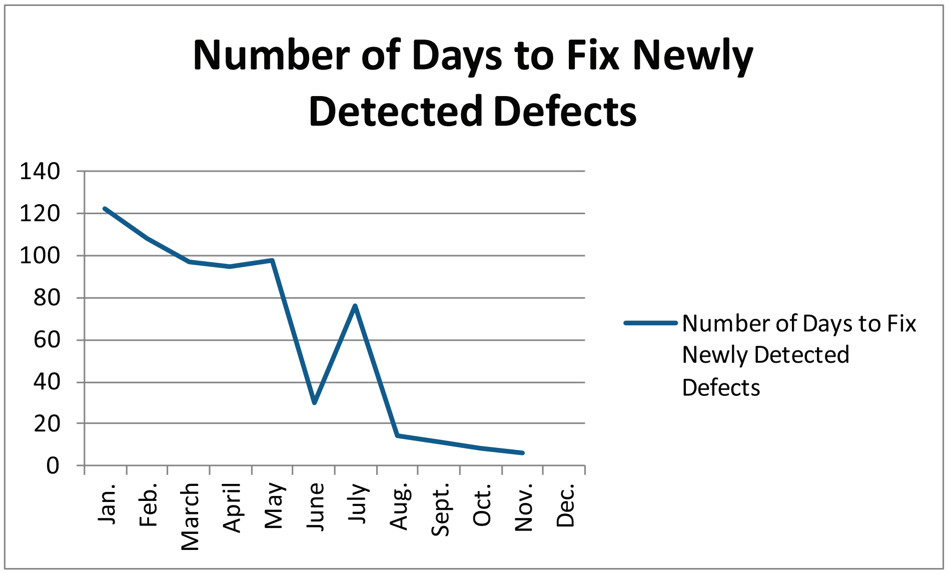

Table 13: Number of Days to Fix Newly Detected Defects

In addition to focusing on new defects, Dave also ensured that critical legacy defects were steadily addressed. The following chart shows the overall number of defects that were fixed in 2013. The chart shows a spike in the number of defects fixed that occurred when Dave commenced his role as the project administrator within Coverity Scan. The dip in December is a reflection of a temporary lull in activity due to the holiday season.

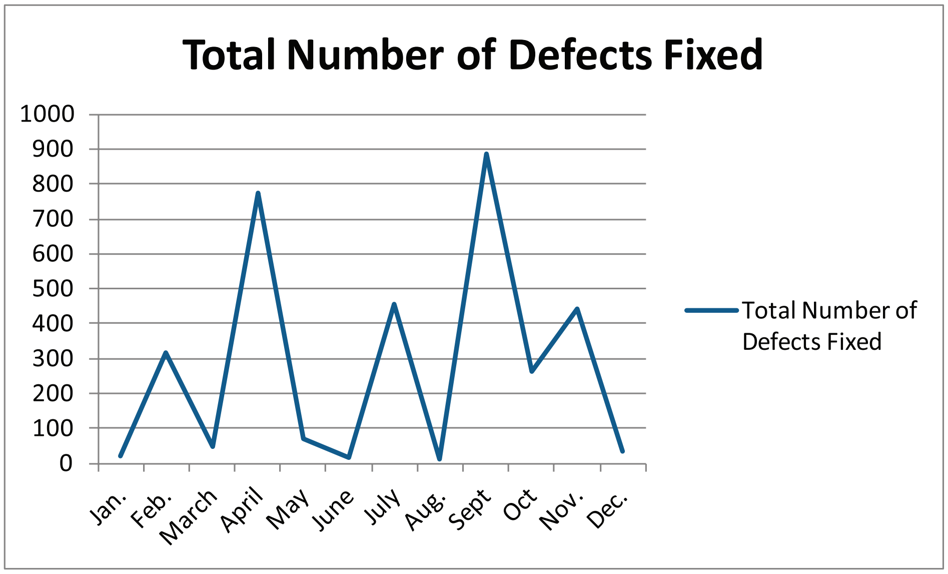

Table 14: Total Number of Defects Fixed

An Interview with Dave Jones: Coverity Scan Administrator for Linux and Blogger

Q: *What has your overall experience with the Coverity Scan service been like?*

A: Over time, the defect rate is decreasing. For every point release of Linux the defect rate drops by around 200 or so bugs. In the beginning, it was about 5.4k issues, then down to 5.2, now down to 5k.

Overall there is an increased interest in the bugs that are being found through the Coverity Scan service. We have more people looking at the bugs and taking them more seriously.

Q: *As you know, we recently added an observer role to the Scan service. How do you see this benefitting your project?*

A: Just having exposure to the project is useful because it does increase participation.

We would like to see observers have access to old bugs in the system as it is useful to have newcomers to a project exposed to actual bugs, especially some of the older bugs that perhaps haven't been getting attention because the original developers have moved on to other code.

For many new developers, they have not dealt with a body of code of this size and complexity, and some of the bugs scan finds are not always obvious. Even to seasoned developers, sometimes it can take a lot of thought to figure out if the bug can actually occur.

Q: *As a long-time participant in the Scan service, what have been the greatest benefits the Linux project has received?*

A: When I first signed up for Scan last July, I was horrified by the more than 5,000 defects I saw. However, implementing daily scans has been really useful. We're now seeing some bugs fixed within 24 hours of being introduced, which means those bugs aren't ending up in a release and users are never seeing them.

Q: *How has your use of the service evolved?*

A: In the past, we had about 12 members of the project who had access to the Scan database. They would log in every once in a while and be a bit overwhelmed by the number of defects. There was awareness of the Coverity Scan service but not lots of momentum. Our goal behind moving to regular builds was to generate more interest and drive adoption. People are more interested and inclined to fix fresh issues while the code changes are still in their short term memory than they are to look into issues from several years earlier.

The only realistic way to approach the bug fix is to try and keep up with the new issues as they are detected, while slowly chipping away at the existing backlog of older issues. Often times with old bugs, we don't fix them directly. The code often gets rewritten for unrelated reasons, thus eliminating the defect. In the Linux project, some code has a very high turnover rate.

Q: *Are quality and security of equal importance?*

A: A security problem is something people take seriously but at the same time quality can be very important. I would say most kernel developers treat quality and security on equal footing. We take the same approach with both as it's often too much work to try to separate the two we just try to focus on fixing the critical bugs. We do treat some of the checkers within Coverity with higher importance.

Q: *Do you have roles in place to ensure that high-quality and secure code is being produced?*

A: The rate of change in Linux is extremely high. [see https://www.linuxfoundation.org/sites/main/files/publications/whowriteslinux.pdf] One of the largest problems Linux faces is getting enough code review. We have considerably more people writing new code than reviewing code, which is one reason tools like Scan are so useful. The Linux development process involves code being passed through multiple trees before it eventually finds its way into Linus Torvalds tree. At each step of the way, numerous people are performing testing on that new code, so by the time it gets merged it usually is in far better shape than it was when it was first submitted.

Q: *What happens if a contributor creates code that does not meet your standards for quality or security?*

A: Ideally, such code should never make it to the stage where it gets into Linus' tree, but it does happen on occasion, when there are insufficient people reviewing new code.

Key Differences: Java and C/C++

In the Coverity Scan service, we are still in the early days of working with Java projects. The service has been open to these projects for less than one year but we've already observed some key differences between Java and C/C++ projects. For C and C++ projects, the developers fixed 46% of all of the resource leaks that were identified. However, for Java projects, only 13% of the resource leaks were fixed. This begs the question as to whether or not Java developers have become overly reliant on some of the built-in protections in the language, such as the garbage collection, which is supposed to protect against memory leaks by returning memory to the memory pool by destroying objects that no longer have a reference to them. Garbage collection can be unpredictable and lead to resource starvation and program crashes. Plus, garbage collection cannot address system resources such as files and sockets so developers should be mindful of this type of issue. We expect to see that as Java projects become more mature in their use of the Coverity Scan service, they will begin to address more of the resource leaks in their projects.

Overall defect density in Java projects was 2.72, which is significantly higher than the .59 defect density of C/ C++ projects in 2013. There are several factors that impact the density level. First, the analysis algorithms for Java and C/C++ differ. The analysis we provide for the Scan service includes the popular FindBugs checkers, which are very useful. Many of the FindBugs checkers generate large quantities of results, in particular in the areas of dodgy code, performance and bad practices. Another factor to consider when assessing the defect density of the Java programs is the length of time the Scan service has been available to the projects. It is not uncommon to see higher defect density rates in the early stages of a project's participation in the service, followed by a reduction in the defect density over time.

TABLE 15: DEFECTS FIXED BY TYPE AND IMPACT IN JAVA 2013

Category	Quantity	Impact
Null pointer dereferences	546	Medium
FindBugs*: Dodgy code	423	Low
FindBugs: Performance	355	Low
FindBugs: Bad practice	282	Low
FindBugs: Internationalization	273	Low
Concurrent data access violations	211	Low
Resource leaks	143	High
Class hierarchy inconsistencies	123	Medium
FindBugs: Correctness	107	Medium
Control flow issues	61	Medium
FindBugs: Multithreaded correctness	49	Medium

Exceptional resource leaks	39	Low
Integer handling issues	25	Medium
API usage errors	13	Medium
Error handling issues	9	Medium
Incorrect expression	7	Medium
Program hangs	4	Medium
FindBugs: Malicious code vulnerability	3	Low
Performance inefficiencies	2	Low
Total	2,691	

**Note: all category types preceded by FindBugs indicates that this is a FindBugs checker. All others are Coverity analysis algorithms.*

TABLE 16: DEFECTS OUTSTANDING BY TYPE AND IMPACT IN JAVA AS OF DEC. 31, 2013

Category	Quantity	Impact
FindBugs: Dodgy code	5,715	Low
Null pointer dereferences	4,743	Medium
FindBugs: Performance	3,492	Low
FindBugs: Bad practice	2,762	Low
FindBugs: Internationalization	2,062	Low
Resource leaks	1,000	High
Concurrent data access violations	986	Low
FindBugs: Multithreaded correctness	555	Medium
FindBugs: Correctness	482	Medium
Exceptional resource leaks	324	Low
Class hierarchy inconsistencies	319	Medium
Control flow issues	295	Medium
Error handling issues	120	Medium
Integer handling issues	76	Medium
API usage errors	60	Medium

Incorrect expression	44	Medium
FindBugs: Malicious code vulnerability	44	Low
Program hangs	22	Medium
Performance inefficiencies	11	Low
FindBugs: Security	2	Medium
Total	23,113	

Java Project Snapshots

Apache Cassandra, Apache CloudStack, Apache Hadoop and Apache HBase and are just a few of the more than 100 Java projects that joined the Coverity Scan service in 2013. The following is a snapshot of the outstanding and fixed defects for these key projects.

Apache Cassandra

Apache Cassandra (Cassandra) is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. It offers robust support for clusters spanning multiple datacenters with asynchronous masterless replication, allowing low latency operations for all clients. Cassandra joined the service in February of 2013 and the first analysis run was conducted in July of 2013. It has more than 345,000 lines of code and developers fixed 86 defects in 2013.

TABLE 17: APACHE CASSANDRA 2.0 DEFECTS FOUND AND FIXED BY TYPE AND IMPACT IN 2013

Category	Outstanding	Fixed	Impact
Null pointer dereferences	53	14	Medium
Resource leaks	48	7	High
FindBugs: Bad practice	42	2	Low
FindBugs: Dodgy code	37	6	Low
FindBugs: Performance	27	2	Low
FindBugs: Internationalization	18	0	Low
FindBugs: Multithreaded correctness	6	2	Medium
Concurrent data access violations	5	2	Medium
API usage errors	5	0	Medium

Control flow issues	5	0	Medium
Performance inefficiencies	27	0	Low
Integer handling issues	2	1	Medium
Incorrect expression	1	0	Medium
FindBugs: Correctness	0	2	Medium
FindBugs: Malicious code vulnerability	1	0	Low
Grand Total	**252**	**38**	**Defect Density: 1.95**

Apache CloudStack

Apache CloudStack (Cloudstack) is designed to deploy and manage large networks of virtual machines, as a highly available, highly scalable, Infrastructure-as-a-Service (IaaS) platform. It is used by a number of service providers to offer public cloud services, and by many companies to provide an on-premise (private) cloud offering, or as part of a hybrid cloud solution. CloudStack joined Coverity Scan in October of 2013.

TABLE 18: APACHE CLOUDSTACK DEFECTS FOUND BY TYPE AND IMPACT IN 2013

Category	Outstanding	Fixed	Impact
FindBugs: Dodgy code	2,617	49	Low
Null pointer dereferences	1,831	37	Medium
FindBugs: Performance	1,744	34	Low
FindBugs: Bad practice	215	28	Low
FindBugs: Internationalization	136	8	Low
Resource leaks	126	13	High
Exceptional resource leaks	104	9	Low
FindBugs: Correctness	87	14	Medium
Concurrent data access violations	43	1	Medium
FindBugs: Multithreaded correctness	34	1	Medium
Control flow issues	25	4	Medium

Integer handling issues	10	2	Medium
Error handling issues	8	0	Medium
Class hierarchy inconsistencies	6	4	Medium
Incorrect expression	6	0	Medium
API usage errors	2	0	Medium
Grand Total	**6,994**	**220**	**Defect Density: 6.96**

Apache Hadoop

The Apache Hadoop (Hadoop) software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer. The project joined the Coverity Scan service in August of 2013. Hadoop has more than 320,000 lines of code in their core components. Since that time developers have found and fixed the following defects in the code:

TABLE 19: APACHE HADOOP DEFECTS OUTSTANDING AND FIXED BY TYPE AND IMPACT IN 2013

Category	Outstanding	Fixed	Impact
FindBugs: Dodgy code	128	3	Low
Null pointer dereferences	100	20	Medium
Concurrent data access violations	84	5	Medium
FindBugs: Internationalization	65	9	Low
FindBugs: Bad practice	33	2	Low
Resource leaks	30	1	High
FindBugs: Multithreaded correctness	6	2	Medium
Class hierarchy inconsistencies	3	1	Medium
FindBugs: Correctness	2	2	Medium
API usage errors	2	0	Medium
Incorrect expression	2	0	Medium

Integer handling issues	2	0	Medium
Performance inefficiencies	1	0	Low
Program hangs	1	1	Medium
Error handling issues	1	0	Medium
Grand Total	**457**	**46**	**Defect Density: 1.71**

Apache HBase

Apache HBase (HBase) is the Hadoop database. It is a distributed, scalable, big data store and a sub-project of the Apache Hadoop project. It is used to provide real-time read and write access to big data. According to The Apache Software Foundation, the primary objective of HBase is the hosting of very large tables (billions of rows X millions of columns) atop clusters of commodity hardware. They joined the Coverity Scan service in late August of 2013. The project has 487,803 lines of code and in the short time it has been using the Scan service, developers have fixed 220 defects including 66% of all resource leaks.

TABLE 20: APACHE HBASE DEFECTS FOUND AND FIXED BY TYPE AND IMPACT IN 2013

Category	Outstanding	Fixed	Impact
Category	Outstanding	Fixed	Impact
Null pointer dereferences	98	44	Medium
Concurrent data access violations	82	18	Medium
FindBugs: Bad practice	55	25	Low
FindBugs: Internationalization	49	36	Low
FindBugs: Dodgy code	37	14	Low
FindBugs: Multithreaded correctness	30	6	Medium
Resource leaks	21	41	High
Control flow issues	8	6	Medium
Exceptional resource leaks	6	5	Low
Integer handling issues	6	1	Medium
API usage errors	5	2	Medium
Error handling issues	3	0	Medium
Class hierarchy inconsistencies	3	0	Medium

Incorrect expression	1	2	Medium
FindBugs: Malicious code vulnerability	4	0	Low
FindBugs: Performance	1	8	Low
FindBugs: Correctness	1	9	Medium
Program hangs	0	1	Medium
Performance inefficiencies	0	2	Low
Grand Total	**410**	**220**	**Defect Density: 2.33**

Conclusion and Next Steps

The year 2013 was an exciting one for the Coverity Scan service. We saw unprecedented levels of growth and adoption by open source users. We are excited about adding the Project Observer role to the service, making it even easier for users to see the high-quality of the participating open source projects. We have many additional enhancements planned for 2014, which we look forward to communicating in the coming months. We would like to thank all of our Scan members and the open source community at large for their interest and support of the Coverity Scan service. If you would like to register a new project, contribute to or observe an existing project, visit us at https://scan.coverity.com/

About Coverity

Coverity, Inc., a Synopsys company (Nasdaq:SNPS), is a leading provider of software quality and security testing solutions. Coverity's award-winning development testing platform helps developers create and deliver better software, faster, by automatically testing source code for software defects that could lead to product crashes, unexpected behavior, security breaches or catastrophic system failure. The world's largest brands rely on Coverity to help ensure the quality, safety and security of their products and services. For more information, visit www.coverity.com, follow us on Twitter or check out our blog.