coverity®

Coverity Scan:

2012 Open Source Report

Table of Contents

Coverity Scan: A Brief Introduction 2

Coverity Scan: Then and Now 4

Linux: Then and Now 15

Development Testing in Action 19

Conclusion and Next Steps for Coverity Scan 29

About Coverity 30

Appendix A: Coverity Software Integrity Report for AMANDA 31

Appendix B: Coverity Software Integrity Report for ffmpeg 36

Appendix C: Coverity Software Integrity Report for Linux 41

Appendix D: Coverity Software Integrity Report for Mesa 46

Appendix E: Coverity Software Integrity Report for NTP 51

Appendix F: Coverity Software Integrity Report for XBMC 56

Coverity Scan: A Brief Introduction

For those not familiar with the Coverity Scan™ service, it began as the largest public-private sector research project in the world 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 Scan service enables open source developers to scan–or test–their 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. (More than 20,000 defects identified by the Coverity Scan service were fixed by open source developers in 2012 alone!) The Scan service is powered by our award-winning Coverity SAVE® static analysis verification engine, which applies multiple patented techniques for highly accurate defect detection. Coverity SAVE is based on more than a decade of research and development, and analysis of more than five billion lines of proprietary and open source code. The intelligence we gather into a variety of source code across both open source and commercial projects is used to continually tune the SAVE algorithms, which broadens the range and depth of defect detection, to continually improve software quality.

Since the introduction of the Coverity Scan service in 2006, Coverity has worked with more than 300 of the most widely adopted C/C++ open source projects, including Linux, PHP and Apache. In March 2013, we expanded the service to include Java open source projects and have begun working with Hudson Server, Eclipse Code Recommender and Apache Cassandra, among others. We also joined the Eclipse Foundation and now offer a Hudson plugin that integrates with projects hosted by the Eclipse Foundation so they can access the Scan service in their development workflow. We welcome our new friends in the Java open source development community and look forward to working with more projects in the future.

Our mission with the Coverity Scan service is to help further the adoption of open source software by providing open source projects with an opportunity to build code quality and security into their development process in the same way as commercial projects. We continually learn from our commercial customers and are building this knowledge into our technology—from improved accuracy to ease-of-use in the development workflow—to bring these best practices to the open source community. In addition, we are seeing more commercial projects build policy management and governance into their development processes as a way to create a common and objective baseline to measure code quality across both internally developed and third party software (including open source). Based on the millions of lines of open source code we have scanned over the past seven years, we have become an authority on the state of open source software quality. Our aim is to provide the industry with an objective industry standard that commercial projects can use to assess open source software quality, and ultimately increase the adoption of open source.

We are proud to provide a valuable service that helps open source developers write and deliver high-quality, secure code, and thank the open source community for its ongoing commitment to the Coverity Scan service and the continuous improvement of open source software quality. Here are just a few examples of what open source developers are saying about the Coverity Scan service:

"Coverity is a code-analysis tool – an extremely good one, probably at this moment the best in the world."
- GPSd -

"Several other Coverity issues have been resolved and their fixes have made their way into release candidate 7. I'm pretty excited about the results, and have no doubt that Coverity is adding value to our project."
- POV-Ray -

"For those who have either never used static analysis tools, or have simply never used Coverity, don't fall into the trap of thinking that gcc–pedantic–Wall or even LLVM's scan-build should be 'good enough for anyone' – it simply is not. Consider too Steckel's Rule to Success, 'Good enough is never good enough'."
- Upstart and Whoopsie -

"Vulnerability Notifications–We recommend all administrators upgrade immediately. The vulnerability was created in commit...clang-analyzer did not find this issue. However, a Coverity scan did discover it."
- FreeRADIUS -

"Ah, that's cool. Pretty neat that an automated tool can catch mutex lock problems in conditional statements wrapped in macros! I'm impressed."
- Genivi -

"Coverity performs very deep analysis and its results may well surprise you...but rather that than unexpected surprises for your users. Apologies if this post sounds like a bit of a sales pitch. It really isn't though: the Coverity service is free and what they are offering really is too good to ignore."
- Upstart and Whoopsie -

"You have a very good product and provide a great service to the open source community (certainly to the Linux kernel community)."
- Linux -

"Thank you guys for making such an awesome tool accessible to the open source community!"
- Java Developer -

Coverity Scan: Then and Now

The 2012 Coverity Scan report is a milestone, as it's the fifth anniversary of this annual report. As such, we wanted to take a look back over the past five years to see if we noticed any trends or material changes in open source software quality. And what we found is that the more things change, the more they stay the same…

According to the 2013 Future of Open Source Survey[1], open source software has reached a depth and maturity where quality is no longer a barrier to adoption, and in fact is driving companies to increased use of open source software. This trend is reinforced by thousands of developers working to reduce defects in code, improve its security and innovate with new features and enhancements that get closer to what users want.

Overall, open source software projects that join the Coverity Scan service have a quality-first mindset. They have embraced development testing as part of their quality processes through the adoption of technology such as static analysis, to help build and sustain extremely high levels of software quality and security. Software quality for the projects in the Scan service has been exemplary over the last five years, at a sustained level that is above the industry average for good software quality; even with increases in codebase size and significant advances in static analysis technology, which increase the breadth and depth of issue detection, quality has remained at a consistently high level. In fact, sustaining this level of quality amidst these external and technological factors is testament to the fact that open source software quality has improved over the past five years, and we believe has raised the bar for good software quality for the entire industry. Let's take a look at some of the metrics and measurements that lead us to this conclusion.

Project and Developer Adoption

Since 2006, we have worked with more than 300 open source software projects as part of the Coverity Scan service. In 2008, there were 120 active projects that participated in the Scan service. The number of active projects has increased over the past five years to 169 active projects at the end of 2012, and as of the time of this report, there are now 254 active projects.

The active project distinction is an important one, as some projects join the Coverity Scan service but do not remain active, mainly due to a lack of project champions. These champions are responsible for reviewing analysis results on a periodic basis, and regularly working with the developers in their respective projects to triage and fix the identified defects. We see a similar trend in our commercial customer base: successful adoption typically takes one or two people to champion the cause within a development project. At a certain point, internal awareness reaches critical mass and development testing becomes part of the workflow—as developers realize the value it provides, by helping them produce better code without slowing them down or disrupting their process. This adoption, of course, is also dependent on having technology that developers will adopt; it must be accurate, provide actionable information that helps fix issues and integrate seamlessly into the development workflow.

In addition, at the end of 2012, the Scan service had 393 active developers who logged in on a consistently frequent basis to triage and fix defects. As of the time of this report, this number has increased to 600 active developers. However, this number doesn't capture the full awareness and scale of the Scan service, as it only represents the developers responsible for logging in to gather defect reports, which they post to mailing lists and message boards for the open source community at large to fix.

[1] Future of Open Source Survey: North Bridge Venture Partners and Black Duck Software

The explosive growth of the Coverity Scan service over the past year affirms the industry's mass adoption of development testing, and static analysis as a mainstream development testing method and best practice in both the open source and commercial software development community.

For the 2012 Coverity Scan Report, we analyzed the top 118 active, open source software projects which have been participating in the Scan service for at least one year, many of which have been active for three or more years; these projects represented more than 68 million lines of unique code scanned and an average codebase size of 580,000 lines of code in 2012. The projects range in size from slightly less than 10,000 lines of code to nearly eight million lines of code, including three projects with more than seven million lines of code each. This is an upward trend from the 2008 report, where the project size ranged from 6,000 lines of code to slightly more than five million lines of code, with a mean project size of 425,179 lines of code. Since many of the projects in the 2008 Report are still active today, we see this rise in the average open source codebase as a proxy for the continued, growing adoption of open source software in the industry.

As we compared the project distribution percentage from the 2011 to 2012 report, we observed an increase in smaller project adoption of static analysis, another indication of the rising adoption and critical mass of this technology as a development testing method and the commitment to build quality and security in from the earliest stages of the project's maturity. Another observation which is consistent from 2011 to 2012 is that the number of projects in the 500,000 to 1 million range is low as a percentage of the overall project sample. We will come back to this data point later in the report when we discuss defect density by project size, as it could be an indicator of a 'cliff' that open source projects experience when trying to move beyond the initial stages in the project's lifecycle.

TABLE 1: 2012 PROJECT DISTRIBUTION BY CODEBASE SIZE

Size of Codebase (Lines of Code)	Number of Projects (2012)	% of Total Projects (2012)	% of Total Projects (2011)
Less than 100,000	45	38%	22%
100,000 to 499,999	52	44%	44%
500,000 to 1 million	8	7%	16%
More than 1 million	13	11%	18%

Defect Density

We evaluated average defect density across all active projects in the Coverity Scan service, looking at Coverity Scan Report data from 2008 to 2012.

TABLE 2: AVERAGE DEFECT DENSITY ACROSS ALL ACTIVE COVERITY SCAN PROJECTS, 2008-2012

Coverity Scan Report Year	Average Defect Density
2008	.30
2009	.25
2010	.81
2011	.45
2012	.69

It is tempting to look at this data and come to the conclusion that open source software quality has not only gone through ebbs and flows over the past five years, but has actually declined in the past year. However, it is important to note that the technology has undergone significant innovation over the past five years in terms of the sophistication of analysis, the methods for detecting certain types of defects and the detection of new types of defects. For example, in 2008, the Scan service leveraged version 2.4 of the Coverity static analysis technology, which included 12 checkers (or algorithms for finding defect types); in 2012, the Scan service leveraged version 6.5 of the Coverity® static analysis technology, which included approximately 60 checkers and detected more than 25 types of defects. The impact of this more sophisticated technology is that it can identify more defects in a given codebase. We have also historically seen that codebases with a defect density of 1.0 (or 1 defect for every 1,000 lines of code) is considered good quality software. With a range of .25 to .81 over the past five years, we can say with confidence that 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 we believe has raised the bar for what is considered good quality software for the entire industry.

Defect density in this report is measured by the number of defects per 1,000 lines of code, identified by the Coverity SAVE engine. 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" and "medium-impact" categories, and is a measure of confirmed and potential defects that are left in the codebase as of the time of the report.

A Note on False Positive Rates

The number one barrier to adoption of static analysis technology has historically been its analysis accuracy, or the number of false positives in the analysis results. One of the primary reasons for the rapid adoption of Coverity static analysis technology in both open source and commercial software development projects is the high level of analysis accuracy, or low false positive rate. As we scan more code, both open source and proprietary, we learn from the results and continually tune our analysis engine to improve the breadth and depth of defect detection. Low false positive rates result in more confidence in the defect data and less time wasted with defects that aren't real. In the 2008 report we noted the false positive rate across all open source projects analyzed was 13.32%, in 2009 that number fell to 9.8% and in 2012, the number decreased slightly to 9.7%. Coverity has always been focused on building accurate technology as a means to remove the barriers to developer adoption, setting an industry standard false positive rate that is significantly less than most commercial and open source static analysis tools currently available today.

TABLE 3: DEFECT DENSITY BY PROJECT SIZE IN 2012

Lines of Code	Defect Density
Less than 100,000	0.40
100,000 to 499,999	0.60
500,000 to 1 million	0.44
More than 1 million	0.75
Average Across Projects	**0.69**

We also looked at defect density broken out by codebase size. Again, while all projects have above average software quality, the variances across projects could be an indicator of where a project is in its maturity and how that impacts quality. The lowest defect density is with projects with fewer than 100,000 lines of code. During a project's infancy, there is likely to be a more controlled process for software quality and testing; when the codebase size is smaller, it's easier to maintain and coordinate defect fixes. The beginning of a project also tends to attract the "star" developers who want to go above and beyond their day job to contribute to the good of the open source community. They are likely passionate about the type of open source software they are developing—in many cases they are users themselves—and are dedicated to ensuring the project's success. In many open source projects, the developer "pecking order" is largely influenced by the quality of code contributed to the project, thereby rewarding high-quality code development. These factors were common trends noted across the open source projects interviewed in this report.

When a project increases in size to 100,000 to 499,999 lines of code, defect density increases as more defects are introduced and left unfixed. This could be due to many factors, including less control over the codebase, rising complexity of code and more open source developers (with varying levels of skill) contributing to the project. The project may also be at a point where it hasn't matured to build-in the next level of processes, testing and control. This could also be an indicator of a "cliff" in the project's lifecycle, which is illustrated by the significant drop-off in the number of projects with 500,000 to 1 million lines of code.

The 500,000 to 1 million lines of code range is the point in the maturity curve where projects begin to implement additional controls and processes, look at new types of testing methods to implement and have dedicated roles within the community, so there is a single person or team of people who have accountability and enforcement over these processes. This has a positive impact on quality, which is illustrated by the defect density decrease to.44 overall. We see that defect density increases again with projects that have more than 1 million lines of code, but this doesn't mean that the quality of the codebase suffers. These are typically projects that are heavily adopted in the industry, have the backing and support of a commercial company and still have above average software quality. The slight decrease in quality is most likely a function of the codebase size, as well as of the large number of developers who contribute to the project, many of whom are geographically dispersed and operate as a "virtual" development community. With projects of this size, it is also likely that few people know enough of the code to feel comfortable triaging and fixing defects throughout the codebase. In addition, as more people are required to be involved in resolving defects, it takes longer to fix all of the defects. Even harmless defects may require a much larger effort to triage than in a smaller project, leaving them left unfixed for longer periods of time. On the flip side, there are likely more eyes on the code, which leads to more people fixing the defects. We spoke with a member of the Linux community, who described himself as a "janitor" of the codebase. His sole purpose and involvement in Linux is not to write code, but instead to fix identified defects and "clean up after others." These types of projects can also leverage the scale of their development community: another Linux maintainer we spoke with stated that when they need to get developers to fix issues and test code, they just "throw bodies" at the problem. While it is a luxury to have a robust, established development community that can do this, it can also introduce chaos to the process.

Defect Density of Open Source vs. Proprietary Code

The 2011 Coverity Scan Report was the first time we compared open source and proprietary codebase quality, as measured by average defect density, using a sample of anonymous data from Coverity commercial customers. In 2011, we looked at a sample of analysis of more than 300 million lines of code from 41 proprietary codebases and measured a defect density of .64, which was on par with the average defect density of open source codebases of a comparable size. For the 2012 Coverity Scan Report, we looked at a sample of analysis of more than 250 proprietary codebases totaling more than 380 million lines of code, with an average codebase size of nearly 1.5 million lines of code.

TABLE 4: 2012 COMPARISON OF OPEN SOURCE AND PROPRIETARY CODE

Size of Codebase (Lines of Code)	Open Source Code	Proprietary Code
Lines of Code	68,443,361	381,640,186
Number of Projects	118	256
Average Project Size (Lines of Code)	579,944	1,490,782
Defects Outstanding (As of 12/31/12)	47,131	259,029
Defects Fixed in 2012	20,720	208,499
Defect Density	.69	.68

TABLE 5: DEFECT DENSITY BY PROJECT SIZE: OPEN SOURCE VS. PROPRIETARY CODE

Lines of Code	Open Source	Proprietary Code
Less than 100,000	0.40	0.51
100,000 to 499,999	0.60	0.66
500,000 to 1 million	0.44	0.98
More than 1 million	0.75	0.66
Average Across Projects	0.69	0.68

The same analysis we did with the open source software was performed by our customers on the proprietary codebases in our sample. We looked at average defect density in aggregate, as well as by project size. Similar to the 2011 report findings, in 2012 we found the defect density of open source software to be on par with proprietary software. The fact that both open source and commercial projects which have adopted Coverity technology into their development workflow have not only above average software quality as compared to the industry, but mirror each other in terms of quality, is yet another affirmation that the adoption of development testing has raised the bar for what is considered good software quality for all industries that develop software. We recommend that software development teams evaluate any software they are integrating into their project—commercial or open source—to see what types of development testing methods they have implemented as a measure to assess software quality. The fact that a project has adopted development testing is itself an indication of good software quality and code hygiene practices.

While the overall number looks steady across both types of codebases, it's interesting to note that the defect density is higher in proprietary projects with fewer than 100,000 lines of code. However, once a project reaches 1 million lines of code, proprietary codebases eclipse open source codebases in terms of lower defect density. In commercial projects, the beginning stages (fewer than 100,000 lines of code) don't typically receive heavy investment in testing processes and technologies, which could lead to higher defect density and introduction. Also, they may not attract the same level of developer talent as early stage, open source projects. In projects with 500,000 to 1 million lines of code, we see many commercial projects adopt development testing in response to a quality issue, or as part of a company initiative. We also see an increase in implementing governance processes for how code is developed, as well as for acceptance criteria across development teams and suppliers, so when a commercial project reaches 1 million lines of code, it reaps the benefits of these quality measures on a more proactive and preventative basis.

We also evaluated average defect density by industry for the proprietary codebases in our sample. In the 2011 Coverity Scan Report, we broke this out by safety critical and non-safety critical industries. Safety critical includes industries such as aerospace and defense, power and energy, industrial automation, automotive and transportation, and medical devices. Non-safety critical includes industries such as telecommunications and networking, consumer electronics, financial services, software and internet companies, gaming and mobile. In 2011, we found that the average defect density for safety critical industries was .32, while the average defect density for non-safety-critical industries was .68. In 2012 we found the opposite to be true, with the average defect density for safety critical and non-safety critical industries at .82 and .66, respectively. This is not a statement that safety critical software is bad—our customer sample has software quality better than the industry average. However it does signify the rising importance of and commitment to software quality and security in industries such as financial services and mobile, given some of the high-profile software errors and security breaches in the news headlines over the past year.

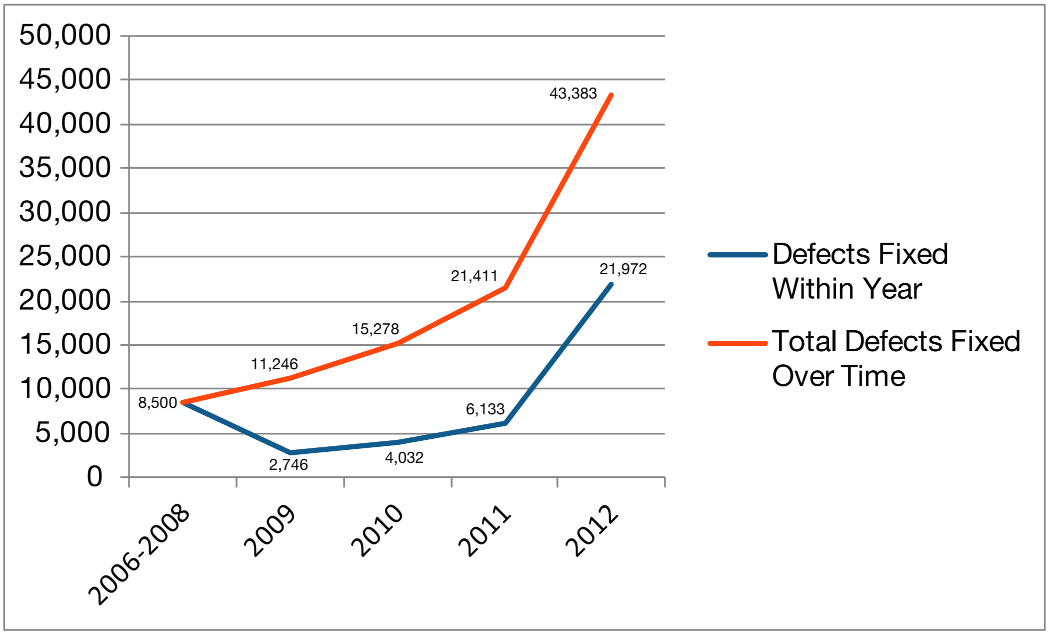
Defects Fixed and Outstanding

Over the course of the past five years, there has been very little change in the types of defects found in open source software; null pointer dereferences and resource leaks have comprised more than 50% of all identified defects.

Defects Fixed

One of the most encouraging trends we are seeing is the increased number of defects fixed since the first Coverity Scan Report in 2008 (which provided a cumulative analysis of defects from 2006 to 2008), both within a given year and cumulatively, over time. While the number of defects introduced has also increased due to more projects, more lines of code analyzed and more advances in the ability to identify defects, the rate by which defects are fixed has not only increased, but at a faster rate than defect introduction. From 2011 to 2012, the lines of code under management in the Scan service increased by 83%, yet the defect fix rate increased by more 250%, from 6,133 defects fixed in 2011 to at least 21,972 defects fixed in 2012. This brings the total number of defects fixed in open source projects through the Coverity Scan service to more than 43,000 defects.

In addition to high false positive rates, another key historical barrier to the adoption of static analysis has been a lack of actionable information on how to fix identified defects. This problem is magnified when it comes to legacy, security static analysis tools: developers are not security experts and need more specific guidance on how to fix security defects. We see the rising defect fix number and rate as a testament to the open source community's ability to easily adopt development testing as part of their daily workflow, as well as their overall dedication to building quality software.



Most Commonly Fixed Defects by Type

The top defects fixed by type is holding steady from 2011 to 2012, with control flow issues, null pointer dereferences and resource leaks in the top three spots. The ranking order based upon quantity fixed by type is slightly varied from 2011, but the same defect categories topped the list consistently for the past two years. For those not familiar with these defect types and their potential impact:

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

- *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, but they are often missed in manual code reviews because they may occur only on certain code paths. If these code paths are never exercised during testing, these defects can be the source of mysterious, unrepeatable crashes in the end user's environment, leading to a need for field support.

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

36% of all defects fixed in 2012 are classified as high-impact (or high-risk) defects. This is consistent with the 2011 Coverity Scan report which reported 35% of defects fixed were high risk in nature. This is not surprising given that these are the most critical types of defects, which could have a significant impact on the system or application if triggered.

TABLE 7: DEFECTS FIXED BY TYPE AND IMPACT IN 2012

Category	Quantity	Impact
Control flow issues	3,464	Medium
Null pointer dereferences	2,724	Medium
Resource leaks	2,544	High
Integer handling issues	2,512	Medium
Memory - corruptions	2,264	High
Memory - illegal accesses	1,693	High
Error handling issues	1,432	Medium
Uninitialized variables	1,374	High
Uninitialized members	918	Medium
Incorrect expression	766	Medium
Insecure data handling	751	Medium
Code maintainability issues	476	Low
API usage errors	257	Medium
Security best practices violations	254	Low
Parse warnings	188	Low
Concurrent data access violations	175	Medium
Program hangs	127	Medium
Performance inefficiencies	49	Low
Class hierarchy inconsistencies	4	Medium

Defects Outstanding

The number of defects outstanding increased from 16,884 at the end of 2011, to 21,972 at the end of 2012, as indicated by the slight increase in average defect density across the open source projects analyzed. If you assumed that every project contained the same number of defects, regardless of size, this would mean that each open source project in this analysis had 399 defects as of the end of 2012, up from 375 defects per project at the end of 2011. Again, this is not a statement that quality is getting worse. Instead, it is proof of the growth of open source codebases and advances in the Coverity static analysis technology in its ability to find more defects.

The top outstanding defects by type are fairly consistent from 2011 to 2012, with null pointer dereferences and control flow issues topping the list both years. However, in 2012, error handling issues moved up, from 5th to 3rd place, in terms of quantity outstanding, while memory corruptions (a high-risk defect type) moved down, from 4th to 8th place. Overall, high-risk defects comprised 26% of all outstanding defects at the end of 2012, which is down slightly from 28% a year earlier. This is a positive indication that open source projects are prioritizing high-risk defect fixes first.

TABLE 8: DEFECTS OUTSTANDING BY TYPE AND IMPACT AS OF DECEMBER 31, 2012

Category	Quantity	Impact
Null pointer dereferences	6,846	Medium
Control flow issues	5,826	Medium
Error handling issues	5,461	Medium
Resource leaks	4,826	High
Memory - illegal accesses	4,299	High
Uninitialized members	3,810	Medium
Integer handling issues	3,791	Medium
Memory - corruptions	3,151	High
Code maintainability issues	1,936	Low
Uninitialized variables	1,692	High
Security best practices violations	1,690	Low
Incorrect expression	1,421	Medium
Insecure data handling	1,372	Medium
API usage errors	359	Medium

Concurrent data access violations	206	Medium
Program hangs	180	Medium
Parse warnings	126	Low
Performance inefficiencies	123	Low
Class hierarchy inconsistencies	9	Medium

Overall, the types of defects most commonly identified by static analysis are easy coding errors to make, even for the most talented and experienced developer. Yet they are very difficult (if not near impossible) to find with other testing methods or manual code review. These are also defects which could lead to a major software failure or security breach if triggered in production or operation. Static analysis is the most effective method to identify these types of critical defects in code, and can no longer be considered just a "nice-to-have" means of testing code for defects.

Linux: Then and Now

Coverity has a long and special history with Linux. In fact, the Linux kernel was the first codebase ever scanned by our static analysis technology in the early 2000's, when Coverity was still a research project in the Computer Science Laboratory at Stanford University. (Additional details regarding Coverity's journey of commercializing static analysis were published in a 2010 article, written by Coverity founders and early stage employees, in *Communications of the ACM*[2].)

Linux subsequently became one of the initial open source projects included in the Coverity Scan service and has been an active project since 2006. In 2006, the current version of Linux was 2.6.16 and the initial scan of Linux was across slightly less than 3.5 million lines of code. Over the course of the past seven years, Linux has had more than 15 releases, the portion of the Linux codebase analyzed by the Scan service has increased by more than 100% and the Linux codebase as a whole has more than doubled.

TABLE 9: LINUX ANALYSIS: 2006 TO 2012

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

Since 2006, the Coverity Scan service has identified 14,804 new defects in Linux, and the Linux community has fixed 8,349 of these defects. It's interesting to note that the number of new defects per year dramatically increased from 2009 to 2010, and again from 2011 to 2012. This is likely due to advances in the Coverity technology that enable detection of a broader range of defects versus a statement of declining quality. Another observation is the dramatic increase in the number of defects fixed by the Linux community from 2010 to 2011, and again from 2011 to 2012, which is a testament to the rising awareness and participation from the Linux community.

[2] 2010, *Communications of the ACM*: A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World

In the 2011 Coverity Scan Report, we recognized Linux 2.6 as one of three projects to be recognized as having superior code quality, and which could be used as an industry benchmark for good software quality, with a defect density of .62 on a codebase of nearly 7 million lines of code. From 2011 to 2012, the Linux codebase increased in size by more than 500,000 lines of code, and the defect density increased slightly to .66. As of the time of this report, Linux version 3.8 is the most recent version, with more than 7.6 million lines of code analyzed via the Coverity Scan service and a defect density of .59. We are happy to report that Linux continues to be a "model citizen" open source project for good software quality.

It is also interesting to compare the defect density of Linux from 2011 to 2012 by component. The most notable quality improvement by component from 2011 to 2012 is in the kernel itself, from .95 to .76. In the 2011 Coverity Scan Report, the Linux kernel had one of the highest defect densities by component.

TABLE 10: LINUX (2011) AND LINUX (2012) DEFECT DENSITY BY COMPONENT

	Linux (2012)	Linux (2011)
Drivers	0.53	0.61
Drivers-Net	0.63	0.61
FS	0.7	0.65
Other	0.9	0.48
Drivers-Staging	0.89	0.97
Networking	0.57	0.67
Sound	0.32	0.38
Kernel	**0.76**	**0.95**
Drivers-ISDN	0.49	0.73
Drivers-SCSI	0	0
ASM	0	0

EXCERPT FROM 2011 COVERITY SCAN REPORT: LINUX 2.6

It's also important to look at the software supply chain within Linux itself. Breaking down the defect density within each of the software components, the kernel has a higher defect density. This is likely because every fix has to be weighed against the risk of destabilizing existing code—it's the "some fixes shouldn't be made until you are changing that area of the code" principle. Also, kernel developers may be reluctant to change code that is known from experience to be stable in the field just to satisfy static analysis results. They may wait until the code is being altered for other purposes to incorporate defect fixes into the new code. On the other hand, the kernel has one of the fewest number of defects classified as high risk compared against other components such as drivers. This is likely due to the criticality and widespread usage of the kernel compared to device drivers, many of which are of interest to only a small portion of the Linux global user base. This may result in a higher tolerance for defects and/or increased effort required in getting the community to commit to inspecting all reported defects over time.

Given the improvement in the kernel defect density from 2011-2012, it is likely that the Linux community took one of the 2012 release cycles as an opportunity to incorporate a larger number of defect fixes to the kernel, as part of the release process.

A Note on the Linux 0day Defect (2009)

In July 2009, a security vulnerability was discovered in the Linux kernel. The issue caught people's attention when Brad Spengler of grsecurity.net, a security advocate and developer, released an exploit for the vulnerability, showing that a user who logged into a Linux system with shell access could bypass system security mechanisms and elevate his access to be equivalent to the system administrator (or "root" user). This is called a 0day defect because the exploit is released with no advance warning that people should patch their systems. So even though the code change to resolve the defect was committed 12 days earlier, on July 5th, many people were operating on vulnerable versions of the code.

Given the sensitivity of security risks in open source software, we do not publicize the specifics of the issues we find through the Coverity Scan service. If and when we do publicly disclose information, we wait until long after patches have been issued and security advisories are published. 0day defects are very disruptive for system administrators, since they demand immediate attention—the consequences of ignoring this type of defect can lead to spending more time to completely reinstall a system when someone uses a 0day and then installs a rootkit (persistent backdoor access).

In this specific case, the actual exploit was complex. It involved a compiler optimization that prevented the code from working as written, and a mechanism for mapping the zero-page of memory, so that a NULL pointer referenced memory under the control of the attacker. Finally, the attacker needed execution to continue past the safety check, which would normally return the POLLERR error code.

The code was added to the Linux kernel on Feb 5th. Specifically, the 'struct sock *sk' line was added. On the face of it, this appeared to be a simple null pointer defect. If the 'tun' pointer returned from __tun_get(tfile) was NULL, then the expression tun->sk would cause a null pointer dereference. For programmers used to working outside of the kernel, the usual result of dereferencing a null pointer is a segmentation violation, and the operating system terminating the application process. When programming in the kernel space, the result is specific to the platform and environment.

The unusual situation in this particular defect was that the misbehavior happened at compile time. gcc looked at the assignment to sk, which used tun as if it was guaranteed to be a valid non-NULL pointer, and decided that there was no point in performing the if (!tun) test below. The if() block was optimized out. It was in the source code, but NOT in the machine code that the compiler output. The programmers included a test to check whether tun was valid, and return an error if it was not. The compiler removed that test, resulting in binaries that would allow processing to continue past the intended checkpoint, even when tun's value was invalid.

This vulnerability involved several coordinated software misbehaviors. Each one on its own might be considered a minor flaw, but the combination provided a way for an attacker to gain inappropriate access to the system.

This defect was identified by the Coverity Scan service on March 31st and made available for viewing by active Linux developers. However, it appears that the bug fix committed on July 5th was not prompted by the Coverity-identified defect.

The Linux development community has fixed thousands of defects identified by the Coverity Scan service, including defects which would have provided additional avenues to exploit this specific defect, among others. We thank the Linux community for its ongoing efforts to build quality and security into the Linux codebase and encourage the Linux developers not yet active with the Coverity Scan service to participate and help reduce the number of outstanding defects to reduce the risk of future security exploits.

This is a concrete example of the types of critical defects identified through the Coverity Scan service, and the consequences they can have if unresolved. Let this be continued motivation to the development community at large to adopt development testing into their daily workflow to promote high-quality, secure software.

Development Testing in Action

As part of the 2012 Coverity Scan Report, we interviewed key contributors within the open source projects at AMANDA, ffmpeg, Mesa, NTP and XBMC—all Scan service projects, with varying sizes of codebases, but with exemplary code quality—to see what types of development testing practices and processes they implement to promote and sustain their high levels of software quality.

AMANDA

AMANDA, otherwise known as Advanced Maryland Automation Network Disk Archiver, is a backup solution that allows an IT administrator to set-up a single master backup server to back up multiple hosts over a network. It was started at the University of Maryland and is now offered via a commercial version, which has been developed by the company Zmanda, a leader in cloud backup and open source backup.

Codebase size: 160,800 lines of code
Defect density: Zero Coverity defects!
Spoke with: Jean-Louis Martineau, Project Gatekeeper, and Paddy Sreenivasan, Developer

Q: *Why do you think you are able to achieve the highest level of code quality?*
A: The number one thing is to have a gatekeeper who enforces a process. All of the patches and changes need to go through this person. Given our project's size, it is easily managed by one person. All of the changes are reviewed. They have extensive tests. We run the tests regularly on regular distributions and make sure it passes before the patch is accepted. We maintain backward compatibility, ensure main web pages are all updated when the changes are made and ensure the documentation is up to date.

Our developers are used to this process—it's been around for some time and they just know to follow it. However, we do have some flexibility in our release schedules; we aren't under pressure to hit a specific ship date, so that helps.

Q: *What happens if you have a developer working on the project who submits code that doesn't meet quality expectations?*
A: It depends on the type of code the person is submitting. Developers submit patches that are specific to particular types of devices, but we don't accept patches unless they make sense. We have a contributor directory where we keep changes that can't be validated. Poor quality is seen as a black mark in our project.

Q: *Do you have people in roles that are designed to help ensure quality (e.g. architect, maintainer)?*
A: Jean Louise is our gatekeeper and acts as the equivalent of Linus for the Linux kernel. We are also trying to shrink the codebase. The software is designed for a specific purpose: backup and recovery. We have device APIs and a core set of APIs so people can add their own plugins. We don't test or vet these—just basic code review.
We have three core developers, plus additional developers as necessary.

Q: *How does development testing fit into your release schedule?*

A: We have a major release once every two years, with minor releases every three to six months. The minor release is mainly used by system administrators, where they submit patches that are specific to their devices. We usually receive two to three patches per month. Some issues will get fixed by others in the community. We use Coverity on a weekly basis, and when we get a Coverity report we make sure to fix all of the issues before the release.

Q: *What other quality testing techniques have you adopted outside of Coverity?*

A: Manual code review. We also use buildbot [another open source tool] to run generic test cases.

Q: *What are the top 3 things about your project that enable you to deliver high quality code?*

A:
- Size of the codebase—it's a manageable size to keep quality under control
- Stability and expertise of the developers—our gatekeeper has been doing this for more than 10 years
- Not having the added pressure of hitting a release date—we don't have to trade-off schedule for quality

Our development process is working well so we aren't planning to change anything. However adding more test cases can always improve quality.

ffmpeg

ffmepg is a cross-platform solution to record, convert, stream and handle audio and video. It includes libavcodec, the leading audio/video codec library. The project has been maintained by Michael Niedermayer since 2004.

Codebase size: 578,967 lines of code
Defect density: .12
Spoke with: Michael, Clément and Stefano (Developers)

Q: *Why do you think you are able to achieve high levels of quality?*
A: We have a lot of users, a testing suite and several people with a lot of experience with the project. We also rely on a variety of automated testing methods (including Coverity) to spot all kinds of random problems, automatically.

Q: *How many developers are working on your project?*
A: We have approximately 10 core developers working on development, maintenance and user support on a daily basis, and 200-300 total contributors per year.

Q: *What happens if you have a developer working on the project who submits code that doesn't meet quality expectations?*
A: If the submission doesn't meet our expected standards, the patch is rejected. If the code is already upstream, it is generally discussed on the mailing list or IRC and we try to find a solution.

Q: *Do you have people in roles that are designed to help ensure quality (e.g. architect, maintainer)?*
A: We have certain people who maintain specific parts of the code—for example, an author/contributor of a particular part of the code will generally review patches related to this code and fix the bugs. Additionally, we have a main maintainer who leads the project (Michael Niedermayer), some developers who follow bugs on our tracker (notably Carl Eugen Hoyos), and some developers who support via the mailing-lists, IRC, etc.

Q: *How do you find your developers?*
A: They come to us to contribute.

Q: *What sort of process do you follow to ensure high quality code?*
A: A patch or patchset is sent to the development mailing list—sometimes it is under a "pull-request" form (e.g. the GitHub model). Any developer can then comment on the patch and after a few iterations, if necessary, it is accepted and pushed into the repository. Generally, Michael Niedermayer has the last word.

Q: *Do you have a formal acceptance criteria?*
A: We have a testing suite which allows the contributor or the committer to run an extensive suite of tests on new code, in order to detect regressions.

Q: *What other quality testing techniques have you adopted outside of Coverity?*

A: We have a testing suite called FATE (FFmpeg Automated Test Environment), where different configurations are proposed and tests are run when there is a new commit. These configurations are maintained by certain developers, who send regular results to the server. Some of the configurations run deep analysis using tools like Address Sanitizer, IOC or Valgrind. There is also a coverage report that enables us to monitor what code is/isn't tested.

Software versioning and history is kept by Git, and we have a ticket manager (Trac) for tracking issues and feature requests.

Q: *What enables you to deliver high quality code?*

A: An extensive community of developers, contributors and users with expertise in several areas. In the event that a defect is introduced into the code, there is a high chance it will be found, reported and eventually fixed by some of our dedicated developers.

Mesa

Mesa is an open-source implementation of the OpenGL API for interactive 3D computer graphics. It runs on multiple platforms, supports a number of hardware GPUs and includes software renderers.

Codebase size: 1,038,075 lines of code
Defect density: .50
Spoke with: Brian Paul, Originator of Mesa (and Engineer at VMware)

Q: *Why do you think you are able to achieve a high level of code quality?*
A: We are meticulous with peer review. Every change that is committed to the code, with few exceptions, is posted and reviewed by people on the mailing list. We have a sign-off process for code review, even before it gets checked in. We have 30-50 active developers on the project, with approximately 15 who I would classify as prolific.

Q: *What happens if you have a developer working on the project who submits code that doesn't meet quality expectations?*
A: We haven't really had cases of people persistently submitting bad code. I started the product 20 years ago and still post my code for review. No one is above the process. The developers involved in the project take pride in their work. It's not just throwing code out to check a box. This code is used by millions of people and the developers take this responsibility very seriously.

Q: *Do you have people in roles that are designed to help ensure quality (e.g. architect, maintainer)?*
A: I am the original author for Mesa. Some may say I'm the captain of the project but I'm just one person—I can't govern the entire project. We have a group of five to seven developers that are the most active, and review posts and code changes to ensure quality. We act as a group of peers.

Q: *How do you recruit developers to your project?*
A: There are corporations who care about Mesa since their products use it as part of their software supply chain, including VMware, Intel and Red Hat, so individuals from those companies are actively engaged with the project to ensure its success. Also, many individual, open-source developers interested in 3D graphics contribute to the project.

Q: *Do you have formal acceptance criteria for code?*
A: We have a governance process for quality, but again, it's just a group of peers who are equally ranked.
There are specific components of the code that developers will pay more attention to. For example, Intel developers focus on supporting its hardware drivers, VMware cares about components relevant to its software, etc.

Q: *What is your release cycle and where does development testing fit into it?*
A: We have a major release every 3 months; we implement bug fixes and release bug fix versions more frequently. We use Coverity every other day as part of our automated build system, and ideally we will get to a point where we run the analysis and report on new defects daily.

Q: *What other quality testing techniques have you adopted outside of Coverity?*

A: We have an open source test suite called piglit which contains approximately 10,000 tests. We also use Valgrind for memory and concurrency issues. Since Mesa is cross platform, we use different compilers which find different, lower level bugs. We use Git as our bug tracking system and repository.

Network Time Protocol (NTP)

NTP is a protocol designed to synchronize the clocks of computers over a network. In operation since before 1985, NTP is one of the oldest Internet protocols in use. NTP was originally designed by David L. Mills of the University of Delaware, who still develops and maintains it with a team of volunteers.

Codebase size: 286,295 lines of code
Defect density: .32
Spoke with: Harlan, Project Manager

Q: *What are your thoughts on recruiting and maintaining high developer talent in your project?*
A: It takes a certain type of person to care about time, and those that care about it are very particular about it. They are very skilled at what they do. These people have well-tuned skills. My advice is to find people who are really good and care about what they do. If they are passionate about the space and the project, they will write high-quality code. For us, finding good developers is incredibly hard. It can take years to find someone good at Windows, Unix and Networking—it's very difficult. We do have a few corporations that care about the space we are in, so they encourage their developers to work on the project but it's on their terms. They care about their own interest in the project so it's not something we can always rely on.

Q: *How do you balance quality with speed?*
A: It is a continuous balance—if there is a need for the development of a specific feature, we can sometimes tolerate lower quality over the short term to get the functionality implemented. We can then go back and improve the quality before we make the production release.

Q: *Do you have people in roles that are designed to help ensure quality (e.g. architect, maintainer)?*
A: We have patch reviews. I look at changes before they are committed to the project and once a change is made to the codebase, there are a handful of people who take a very close look at the changes. Each change we commit also triggers an automatic submission for a Coverity run. We do our best to make sure the code builds on a growing number of different versions of different operating systems. There are certain core files in the NTP distribution that are only changed with the blessing of Dave Mills. We have 2-3 people who are core contributors, and more than 40 other developers who have specific areas of interest in the project.

Q: *Do you have formal acceptance criteria for code?*
A: No.

Q: *What other quality testing techniques have you adopted outside of Coverity?*

A: We use BitKeeper as our SCM system. In addition, unit testing is an increasingly significant part of our quality testing. We usually have Google Summer of Code students work on unit testing suites, but not all developers can be unit test engineers so we are bound by the existing skillset in the project. It's one of our goals to have a sufficiently robust set of these tests; they can serve as examples, so our traditional developers can make tests be part of their patch submissions.

Q: *What are the top things about your project that enable you to deliver high quality code?*

A:

- People are the most important element of the project: having the right skills is key. The people who are working on this project have decades of experience. They know what works.
- Change oversight: even though I have 20 years of experience, if I make a change to the code, there are 3 or 4 people who will run it before committing the code and ping me if they have questions.

XBMC

XBMC is an open source software media player and entertainment hub for digital media.

Codebase size: 1,201,946 lines of code
Defect density: .17
Spoke with: Kyle, Developer (and a Coverity user at his employer)

Q: *Why do you think you are able to achieve a high level of code quality?*
A: To keep high quality for the variety of platforms we support, we are trying to be flexible and test many use cases.

Q: *Describe the developer community on your project.*
A: The team is small and geographically dispersed. We have 20-30 developers with a core group of 5-10 developers who make regular submittals.

Q: *What happens if you have a developer working on the project who submits code that doesn't meet quality expectations?*
A: This happens regularly. Code gets checked into the mainline and then we have to go back and fix issues through bug tracking. This is where Coverity has been very helpful. If a developer has repeated quality issues, they will be threatened to have their merge privileges revoked. There is strong community judgment and quality expectations are high. The developers who contribute higher quality code have more rank and weight in terms of their voice in the community. Developers who have had quality problems in the past will typically take more time to ensure higher levels of quality for future commits.

Our developers take pride in their work. Developers are using this software for themselves on a daily basis, so they are motivated to write high-quality code. The community isn't trying to just check the box.

Q: *Do you have people in roles that are designed to help ensure quality (e.g. architect, maintainer)?*
A: It's a fairly loose process. We don't have specific people who officially look after quality. It's engrained in the developer culture on the project.

Q: *How do you recruit developers for your project?*
A: People find them. People who are interested in the project contribute. It's a meritocracy.

Q: *What is your release cycle?*
A: We have moved to a monthly merge process. We have a one to two week period where the change window is open and then a component testing period, followed by integration testing.

Q: *Do you have formal acceptance criteria for code?*

A: We have a fairly loose process and no formal metrics or targets. We conduct manual code reviews and then at integration into the main trunk, I run Coverity on the code. I try to fix the Coverity defects myself and then post a group of changes for peer review. Developers then comment on the change set.

Q: *What other quality testing techniques have you adopted outside of Coverity?*

A: We use GitHub and Trac for defect tracking. Right now we run the builds manually. We don't have any unit testing targets. We have very good quality given the size of our codebase, but we do see the value in having more formal project management and processes to help maintain quality. Today we don't have any defined development goals other than what individuals want to develop themselves. We need more defined and formalized timelines, goals and metrics.

Conclusion and Next Steps for Coverity Scan

The 2012 Coverity Scan Report reinforces the same conclusion as the past five years have shown: both open source and commercial development projects, regardless of project type, size or industry, that make a commitment to building quality and security into their process through development testing, reap the benefits of high code quality and continue to see quality improvements over time. In fact, we believe these projects have raised the bar for software quality for the entire industry. Given the continually rising complexity of software and speed of development, expanding testing upstream and bringing it into the development process as code is written, in an automated fashion, is no longer a nice to have—it's a must have.

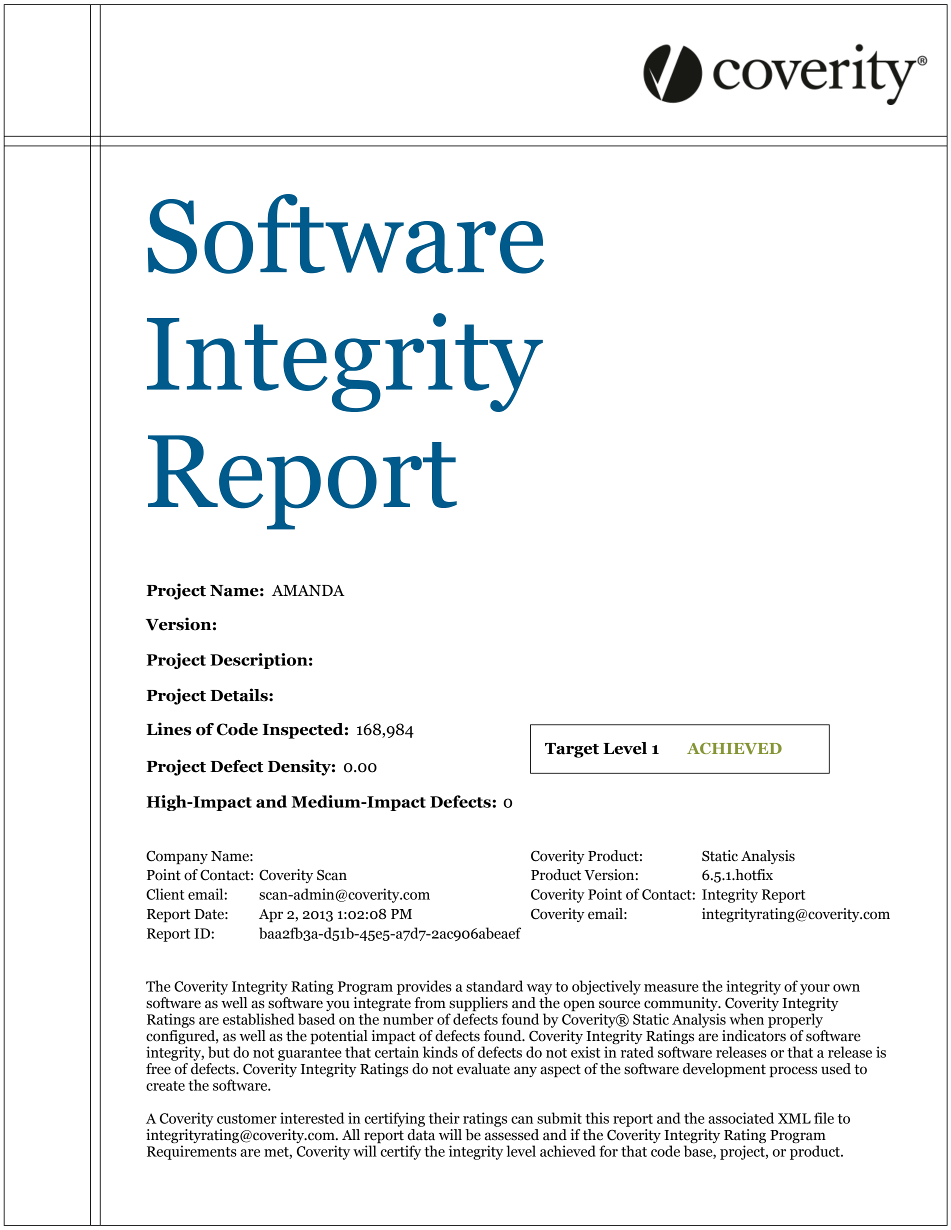
We would like to thank the open source projects participating in the Coverity Scan service for their continued support and participation, and we look forward to extending this service to additional projects in 2013.

About Coverity

Coverity, Inc., (www.coverity.com), the leader in development testing, is the trusted standard for companies that need to protect their brands and bottom lines from software failures. More than 1,100 Coverity customers use Coverity's development testing platform to automatically test source code for software defects that could lead to product crashes, unexpected behavior, security breaches or catastrophic failure. Coverity is a privately held company headquartered in San Francisco. Coverity is funded by Foundation Capital and Benchmark Capital.

- **Appendix A: Coverity Software Integrity Report for AMANDA**
- **Appendix B: Coverity Software Integrity Report for ffmpeg**
- **Appendix C: Coverity Software Integrity Report for Linux 3.6**
- **Appendix D: Coverity Software Integrity Report for Mesa**
- **Appendix E: Coverity Software Integrity Report for NTP**
- **Appendix F: Coverity Software Integrity Report for XBMC**

Appendix A: Coverity Software Integrity Report for AMANDA



coverity®

Software Integrity Report

Project Name: AMANDA

Version:

Project Description:

Project Details:

Lines of Code Inspected: 168,984

Project Defect Density: 0.00

High-Impact and Medium-Impact Defects: 0

Target Level 1 ACHIEVED

Company Name:		Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 2, 2013 1:02:08 PM	Coverity email:	integrityrating@coverity.com
Report ID:	baa2fb3a-d51b-45e5-a7d7-2ac906abeaef		

The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.

coverity®

Software Integrity Report

Project Name: AMANDA

Version:

Project Description:

Project Details:

Lines of Code Inspected: 168,984

Target Level 1 ACHIEVED

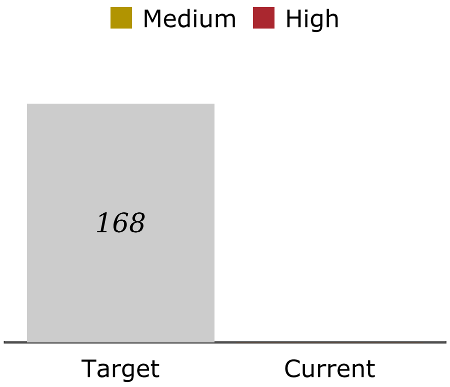
Project Defect Density: 0.00

High-Impact and Medium-Impact Defects: 0

Company Name:		Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 2, 2013 1:02:08 PM	Coverity email:	integrityrating@coverity.com
Report ID:	baa2fb3a-d51b-45e5-a7d7-2ac906abeaef		

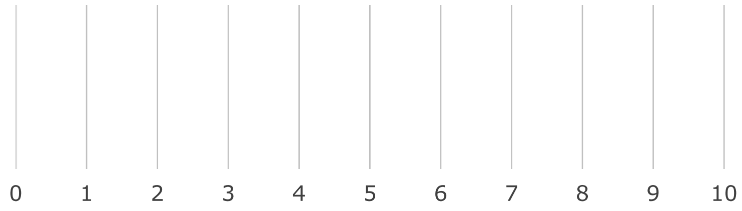
The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.



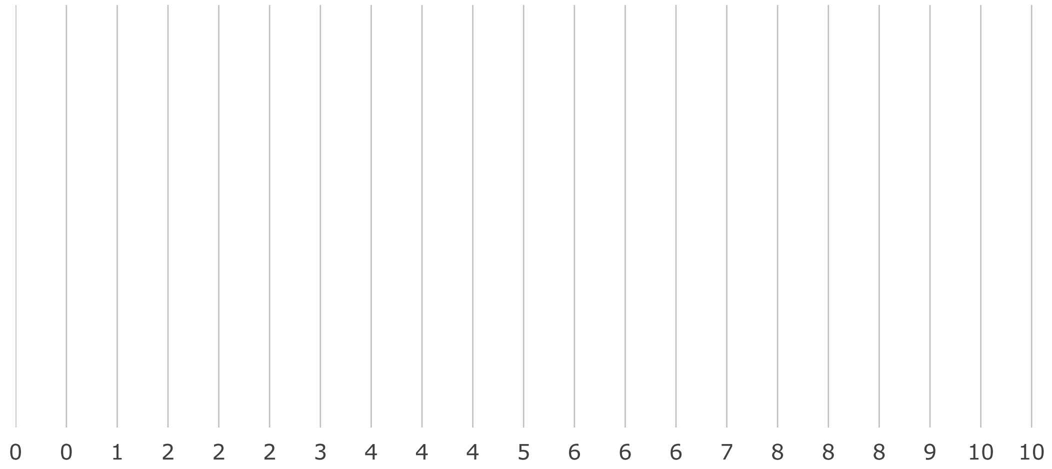
High-Risk Defects

High-impact defects that cause crashes, program instability, and performance problems.



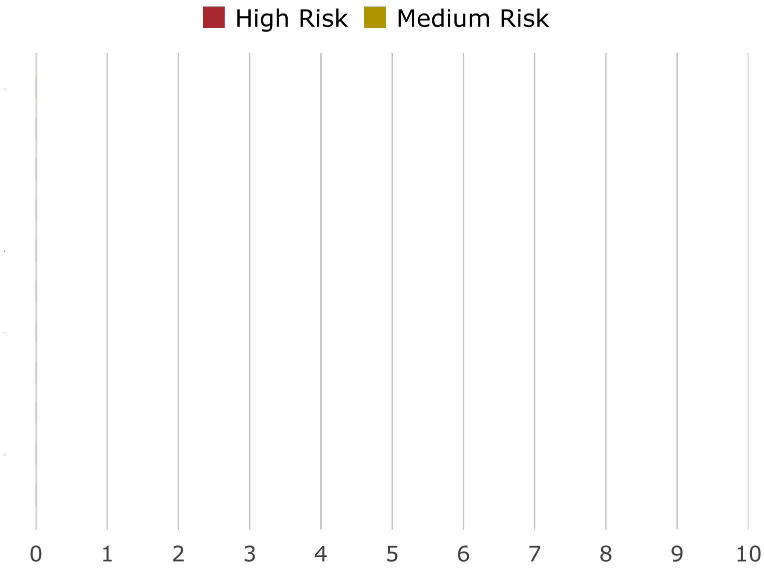
Medium-Risk Defects

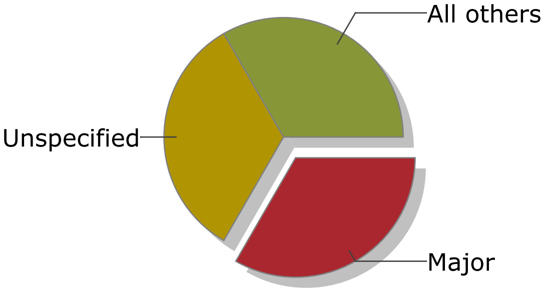
Medium-impact defects that cause incorrect results, concurrency problems, and system freezes.



Defect Risk by Component

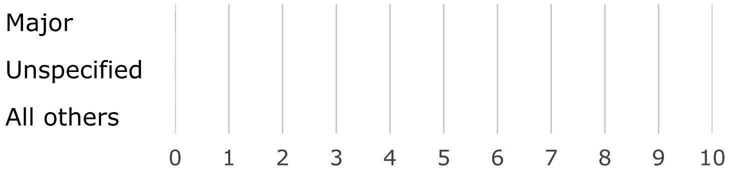
Component	Owner	Defect Density
Other		0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00





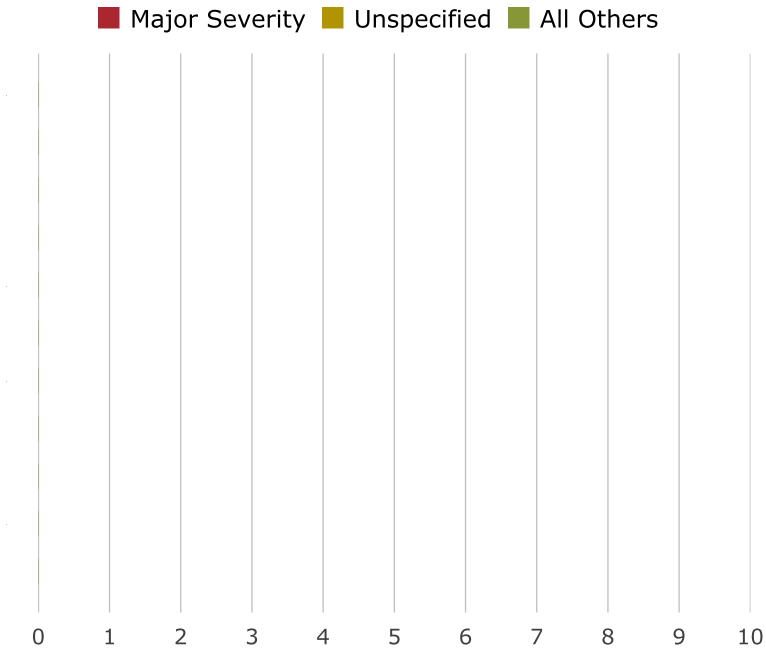
Defects by Assigned Severity

High-severity defects have been tagged by developers as a clear threat to the program's stability and/or security.

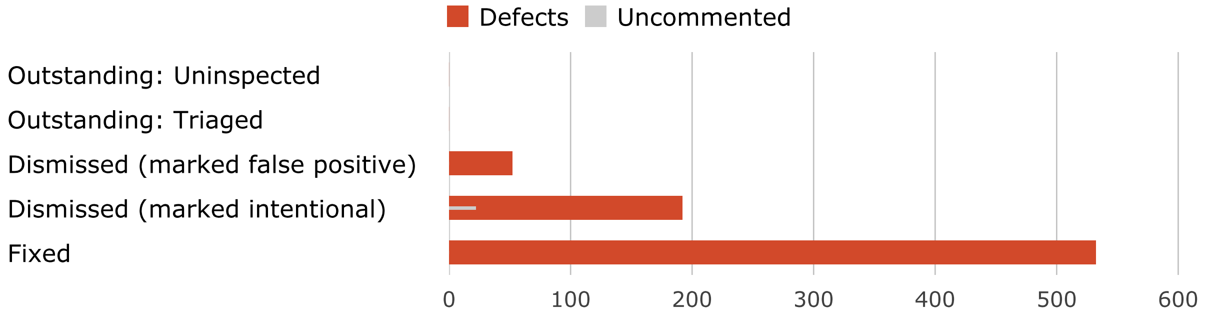


Defect Severities by Component

Component	Owner	Defect Density
Other		0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00



Defects by Triage State



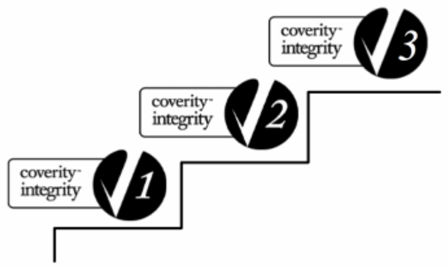
Coverity Software Integrity Report

The Coverity Software Integrity Rating is an objective standard used by developers, management, and business executives to assess the software integrity level of the code they are shipping in their products and systems.

Coverity rating requirements are based on an assessment of several factors:

- Defect density: For a given component or code base, the number of high-risk and medium-risk defects found by static analysis divided by the lines of code analyzed. Defect density excludes fixed defects and defects dismissed as false positives or intentional. For example, if there are 100 high-risk and medium-risk defects found by static analysis in a code base of 100,000 lines of code, the defect density would be 100/100,000 = 1 defect per thousand lines of code.
- Major severity defects: Developers can assess the severity of defects by marking them as Major, Moderate, or Minor (customizations might affect these labels). We consider all defects assigned a severity rating of Major to be worth reporting in the Integrity Report regardless of their risk level because the severity rating is manually assigned by a developer who has reviewed the defect.
- False positive rate: Developers can mark defect reports as false positives if they are not real defects. We consider a false positive rate of less than 20% to be normal for Coverity Static Analysis. A false positive rate above 20% indicates possible misconfiguration, incorrect inspection, use of unusual idioms in the code, or a flaw in our analysis.

Coverity Integrity Level 1 requires the software has less than or equal to 1 defect per thousand lines of code, which is approximately the average defect density for the software industry.



Coverity Integrity Level 2 requires the software has less than or equal to 0.1 defect per thousand lines of code, which is approximately at the 90th percentile for the software industry. This is a much higher bar to satisfy than Level 1. A million-line code base would have to have 100 or fewer defects to qualify for Level 2.

Coverity Integrity Level 3 This is the highest bar in the rating system today. All three of the following criteria need to be met:

- Defect density less than or equal to 0.01 defect per thousand lines of code, which is approximately in the 99th percentile for the software industry. This means that a million-line code base must have ten or fewer defects remaining. The requirement does not specify zero defects because this might force the delay of a release for a few stray static analysis defects that are not in a critical component (or else giving up on achieving a target Level 3 for the release).
- False positive rate less than 20%. If the rate is higher the results need to be audited by Coverity to qualify for this integrity rating level. A higher false positive rate indicates misconfiguration, usage of unusual idioms, or incorrect diagnosis of a large number of defects. The Coverity Static Analysis has less than 20% false positives for most code bases, so we reserve the right to audit false positives when they exceed this threshold.
- Zero defects marked as Major severity by the user. The severity of each defect can be set to Major, Moderate, or Minor. This requirement ensures that all defects marked as Major by the user are fixed because we believe that once human judgment has been applied, all Major defects must be fixed to achieve Level 3.

Level Not Achieved indicates that the target level criteria are not met. This means that the software has too many unresolved static analysis defects in it to qualify for the desired target integrity level. To achieve the target integrity level rating, more defects should be reviewed and fixed.

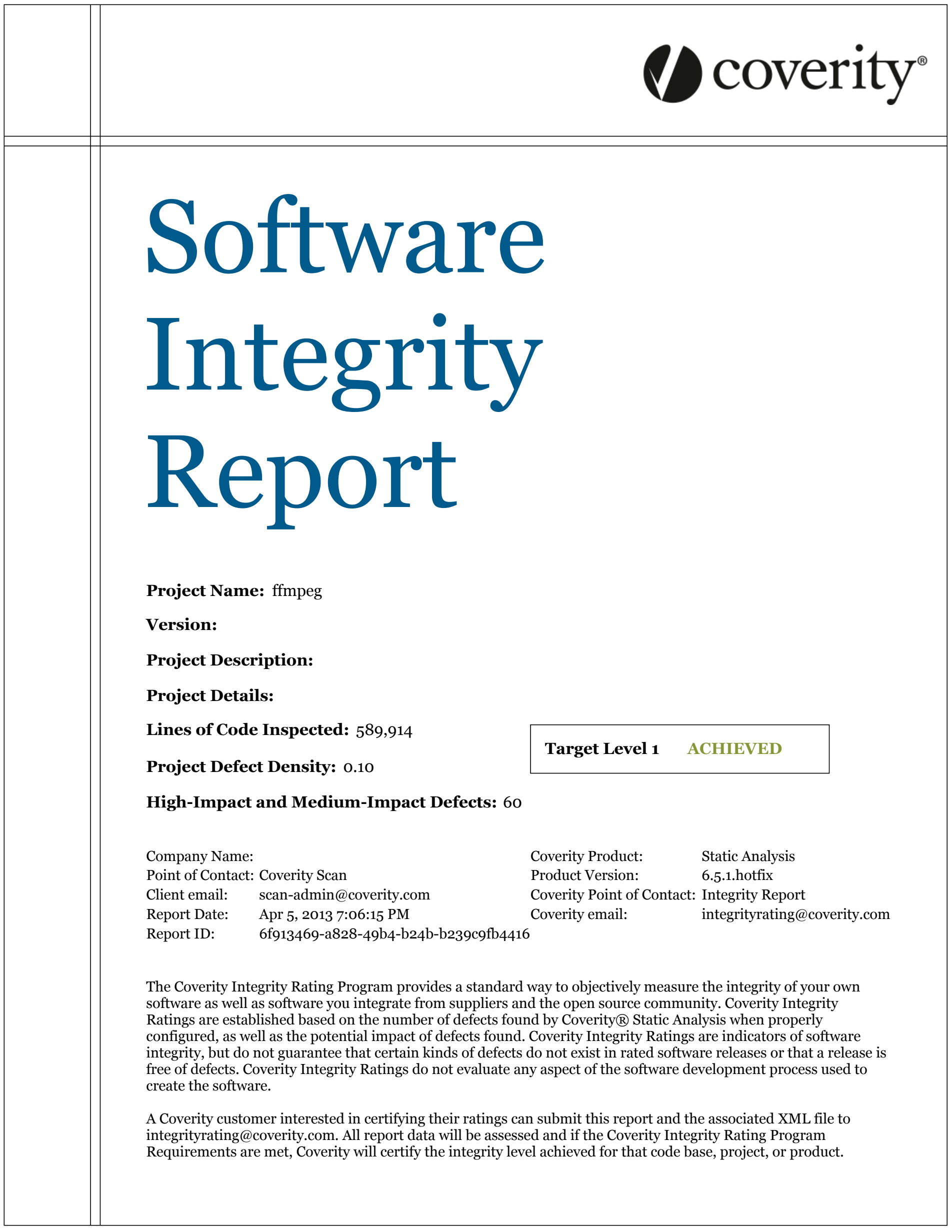
How to Use Your Software Integrity Rating

Set software integrity standards for your projects, products, and teams.
It is often difficult for developers and development management to objectively compare the integrity of code bases, projects, and products. The Coverity Software Integrity Rating is a way to create "apples-to-apples" comparisons and promote the success of development teams that consistently deliver highly-rated software code and products. Development teams can also use these ratings as objective evidence to satisfy requirements for quality and safety standards.

Audit your software supply chain.
It is challenging for companies to assess the integrity of software code from suppliers and partners that they integrate with their offerings. The Coverity Software Integrity Rating is a way to help companies create a common measurement of software integrity across their entire software supply chain.

Promote your commitment to software integrity.
The integrity of your software has a direct impact on the integrity of your brand. Showcasing your commitment to software integrity is a valuable way to boost your brand value. It indicates that they are committed to delivering software that is safe, secure, and performs as expected.

Appendix B: Coverity Software Integrity Report for ffmpeg



coverity®

Software Integrity Report

Project Name: ffmpeg

Version:

Project Description:

Project Details:

Lines of Code Inspected: 589,914

Project Defect Density: 0.10

High-Impact and Medium-Impact Defects: 60

Target Level 1 ACHIEVED

Company Name:		Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 5, 2013 7:06:15 PM	Coverity email:	integrityrating@coverity.com
Report ID:	6f913469-a828-49b4-b24b-b239c9fb4416		

The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.

Software Integrity Report

Project Name: ffmpeg

Version:

Project Description:

Project Details:

Lines of Code Inspected: 589,914

Project Defect Density: 0.10

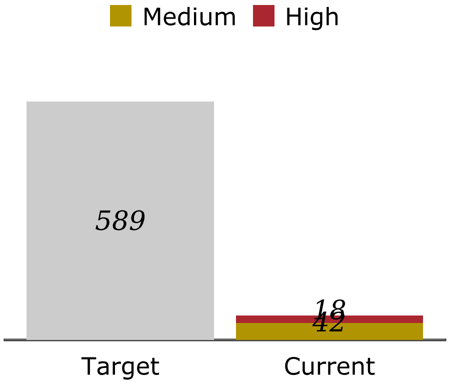
High-Impact and Medium-Impact Defects: 60

Target Level 1 ACHIEVED

Company Name:		Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 5, 2013 7:06:15 PM	Coverity email:	integrityrating@coverity.com
Report ID:	6f913469-a828-49b4-b24b-b239c9fb4416		

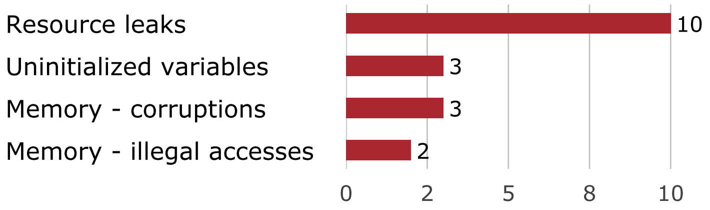
The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.



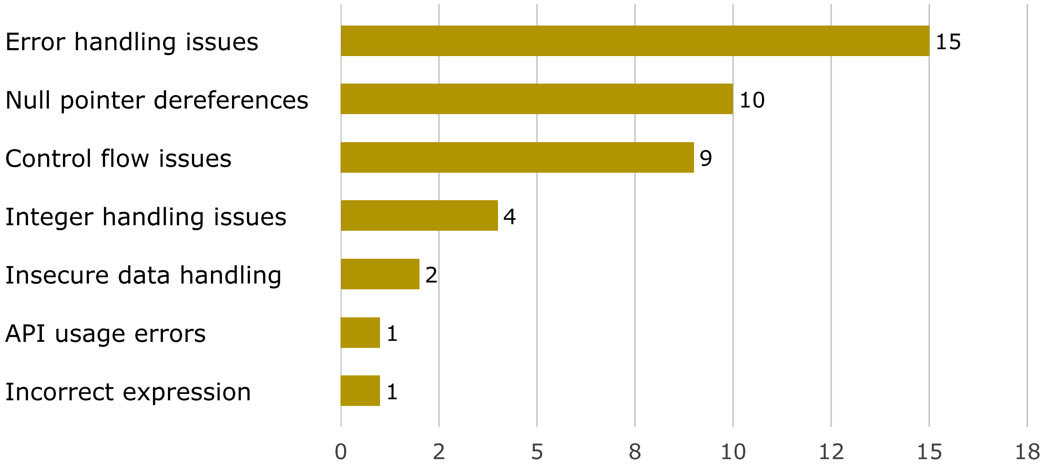
High-Risk Defects

High-impact defects that cause crashes, program instability, and performance problems.



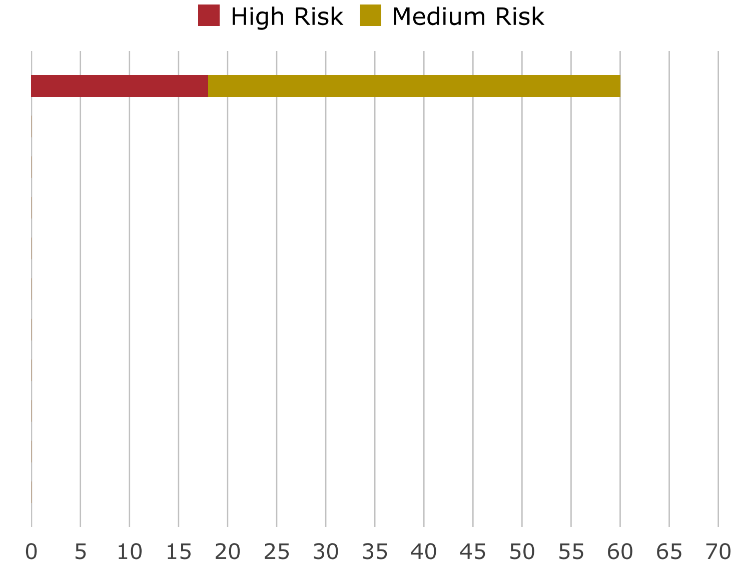
Medium-Risk Defects

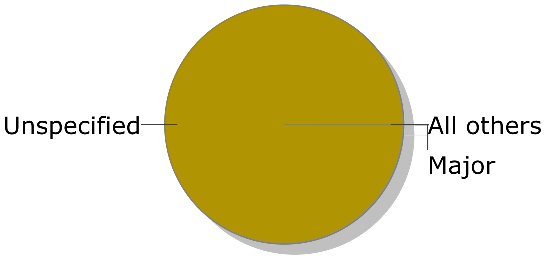
Medium-impact defects that cause incorrect results, concurrency problems, and system freezes.



Defect Risk by Component

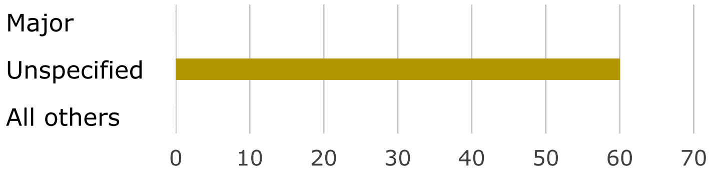
Component	Owner	Defect Density
Other		0.10
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00





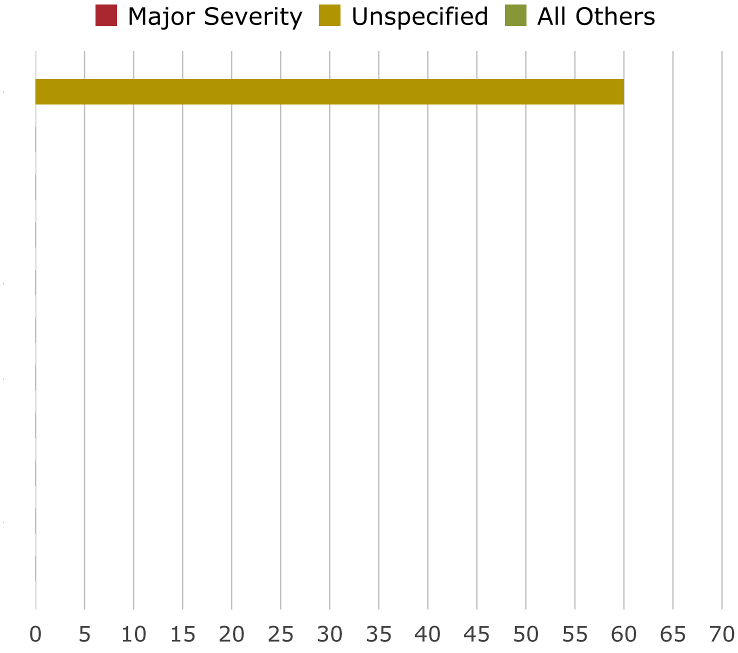
Defects by Assigned Severity

High-severity defects have been tagged by developers as a clear threat to the program's stability and/or security.

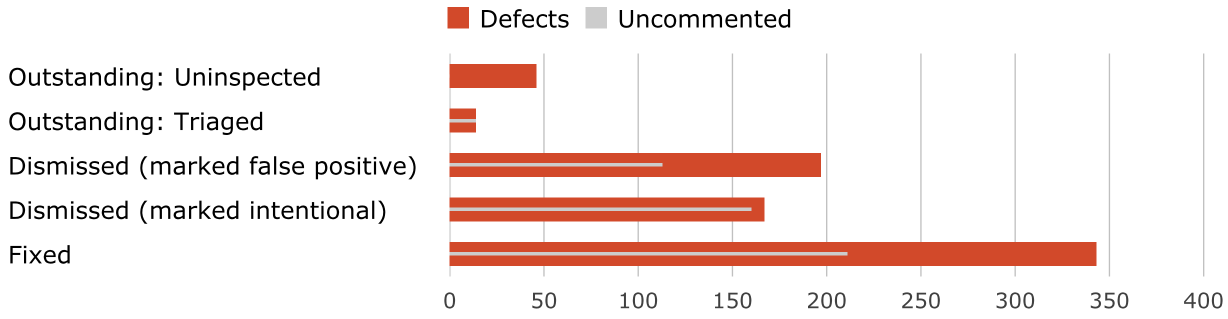


Defect Severities by Component

Component	Owner	Defect Density
Other		0.10
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00



Defects by Triage State



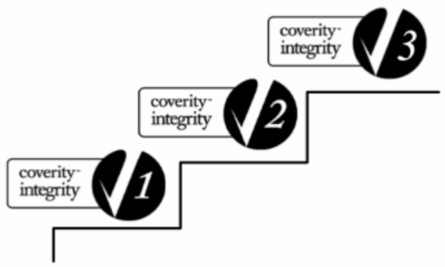
Coverity Software Integrity Report

The Coverity Software Integrity Rating is an objective standard used by developers, management, and business executives to assess the software integrity level of the code they are shipping in their products and systems.

Coverity rating requirements are based on an assessment of several factors:

- Defect density: For a given component or code base, the number of high-risk and medium-risk defects found by static analysis divided by the lines of code analyzed. Defect density excludes fixed defects and defects dismissed as false positives or intentional. For example, if there are 100 high-risk and medium-risk defects found by static analysis in a code base of 100,000 lines of code, the defect density would be 100/100,000 = 1 defect per thousand lines of code.
- Major severity defects: Developers can assess the severity of defects by marking them as Major, Moderate, or Minor (customizations might affect these labels). We consider all defects assigned a severity rating of Major to be worth reporting in the Integrity Report regardless of their risk level because the severity rating is manually assigned by a developer who has reviewed the defect.
- False positive rate: Developers can mark defect reports as false positives if they are not real defects. We consider a false positive rate of less than 20% to be normal for Coverity Static Analysis. A false positive rate above 20% indicates possible misconfiguration, incorrect inspection, use of unusual idioms in the code, or a flaw in our analysis.

Coverity Integrity Level 1 requires the software has less than or equal to 1 defect per thousand lines of code, which is approximately the average defect density for the software industry.



Coverity Integrity Level 2 requires the software has less than or equal to 0.1 defect per thousand lines of code, which is approximately at the 90th percentile for the software industry. This is a much higher bar to satisfy than Level 1. A million-line code base would have to have 100 or fewer defects to qualify for Level 2.

Coverity Integrity Level 3 This is the highest bar in the rating system today. All three of the following criteria need to be met:

- Defect density less than or equal to 0.01 defect per thousand lines of code, which is approximately in the 99th percentile for the software industry. This means that a million-line code base must have ten or fewer defects remaining. The requirement does not specify zero defects because this might force the delay of a release for a few stray static analysis defects that are not in a critical component (or else giving up on achieving a target Level 3 for the release).
- False positive rate less than 20%. If the rate is higher the results need to be audited by Coverity to qualify for this integrity rating level. A higher false positive rate indicates misconfiguration, usage of unusual idioms, or incorrect diagnosis of a large number of defects. The Coverity Static Analysis has less than 20% false positives for most code bases, so we reserve the right to audit false positives when they exceed this threshold.
- Zero defects marked as Major severity by the user. The severity of each defect can be set to Major, Moderate, or Minor. This requirement ensures that all defects marked as Major by the user are fixed because we believe that once human judgment has been applied, all Major defects must be fixed to achieve Level 3.

Level Not Achieved indicates that the target level criteria are not met. This means that the software has too many unresolved static analysis defects in it to qualify for the desired target integrity level. To achieve the target integrity level rating, more defects should be reviewed and fixed.

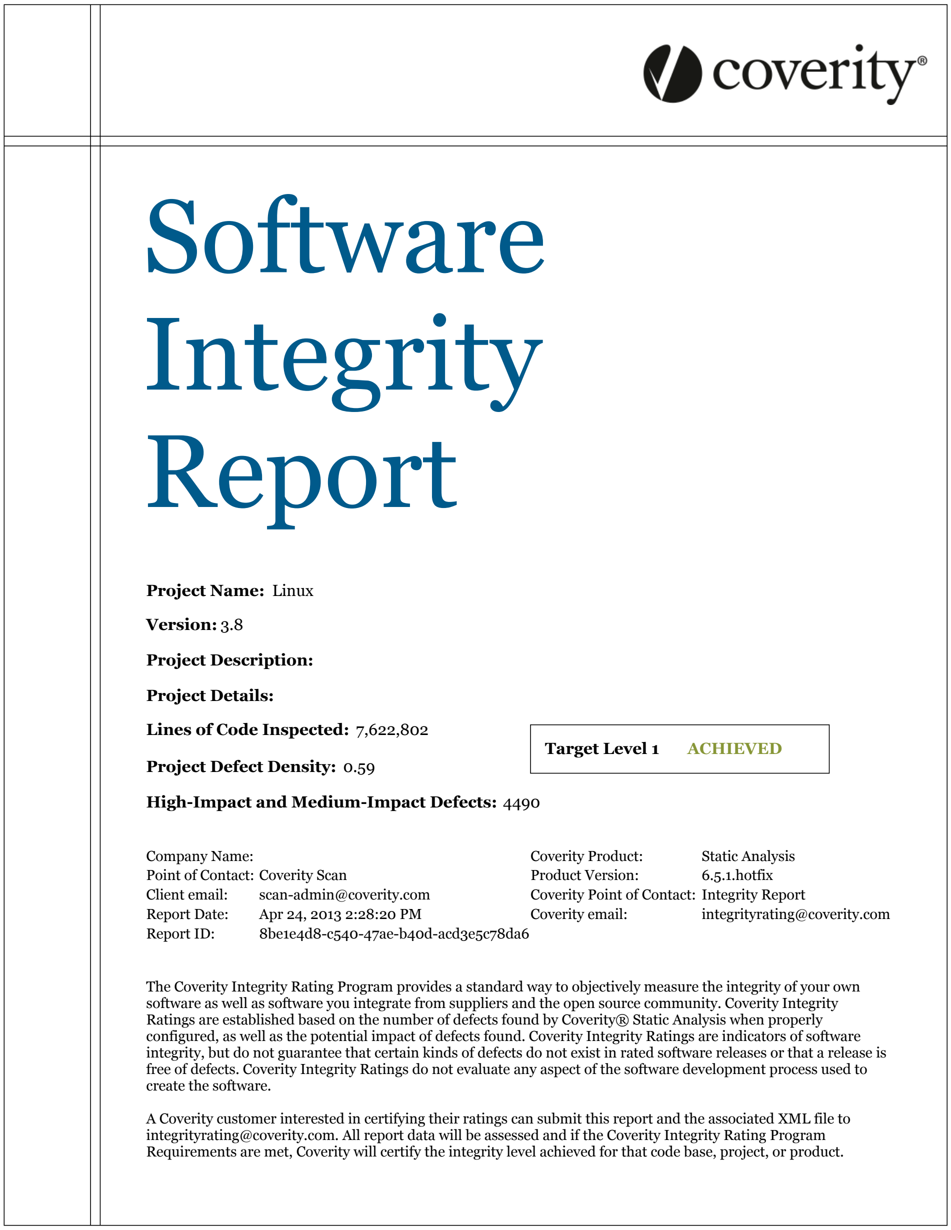
How to Use Your Software Integrity Rating

Set software integrity standards for your projects, products, and teams.
It is often difficult for developers and development management to objectively compare the integrity of code bases, projects, and products. The Coverity Software Integrity Rating is a way to create "apples-to-apples" comparisons and promote the success of development teams that consistently deliver highly-rated software code and products. Development teams can also use these ratings as objective evidence to satisfy requirements for quality and safety standards.

Audit your software supply chain.
It is challenging for companies to assess the integrity of software code from suppliers and partners that they integrate with their offerings. The Coverity Software Integrity Rating is a way to help companies create a common measurement of software integrity across their entire software supply chain.

Promote your commitment to software integrity.
The integrity of your software has a direct impact on the integrity of your brand. Showcasing your commitment to software integrity is a valuable way to boost your brand value. It indicates that they are committed to delivering software that is safe, secure, and performs as expected.

Appendix C: Coverity Software Integrity Report for Linux



coverity®

Software Integrity Report

Project Name: Linux

Version: 3.8

Project Description:

Project Details:

Lines of Code Inspected: 7,622,802

Project Defect Density: 0.59

High-Impact and Medium-Impact Defects: 4490

Target Level 1 ACHIEVED

Company Name:		Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 24, 2013 2:28:20 PM	Coverity email:	integrityrating@coverity.com
Report ID:	8be1e4d8-c540-47ae-b40d-acd3e5c78da6		

The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.

Software Integrity Report

Project Name: Linux

Version: 3.8

Project Description:

Project Details:

Lines of Code Inspected: 7,622,802

Project Defect Density: 0.59

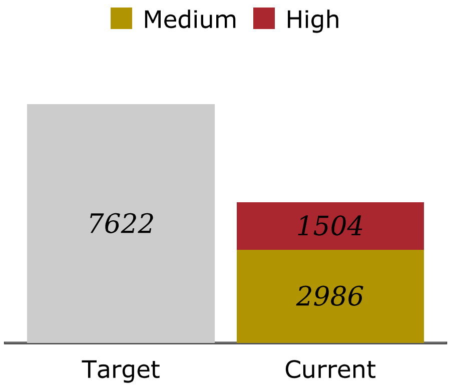
High-Impact and Medium-Impact Defects: 4490

Target Level 1 ACHIEVED

Company Name:		Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 24, 2013 2:28:20 PM	Coverity email:	integrityrating@coverity.com
Report ID:	8be1e4d8-c540-47ae-b40d-acd3e5c78da6		

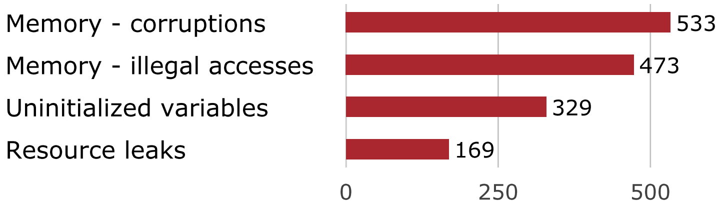
The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.



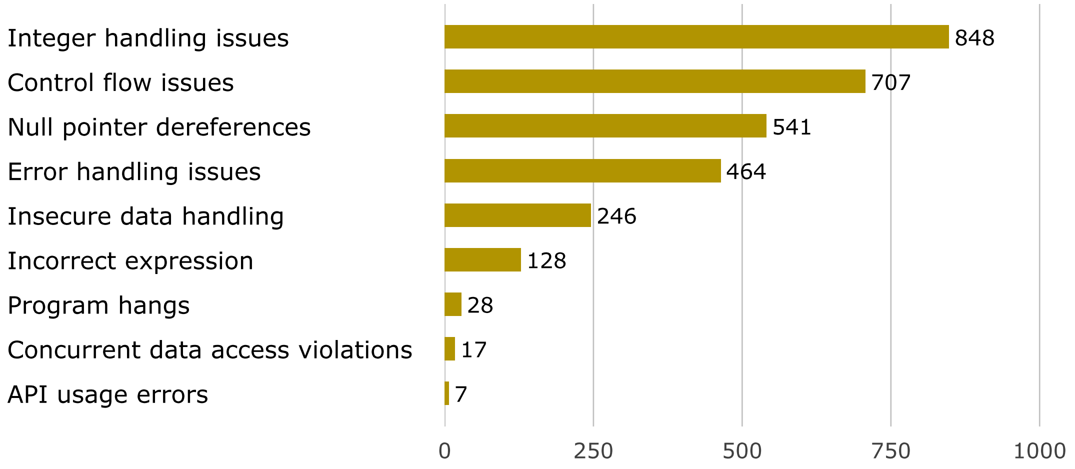
High-Risk Defects

High-impact defects that cause crashes, program instability, and performance problems.



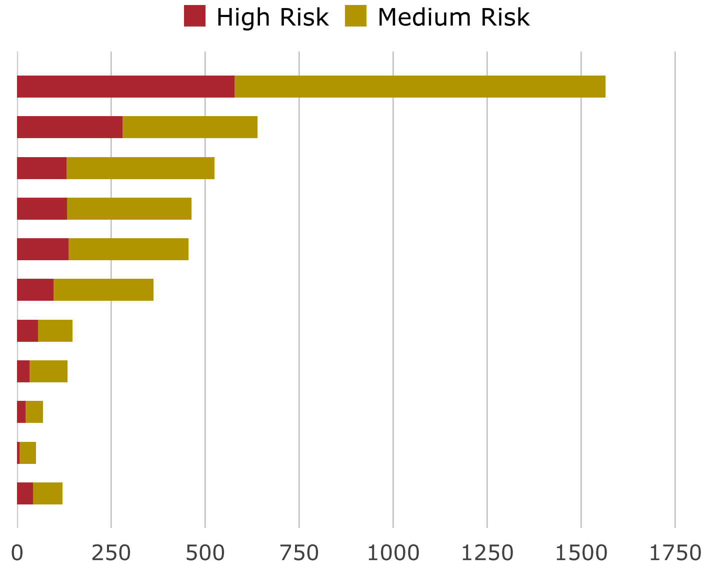
Medium-Risk Defects

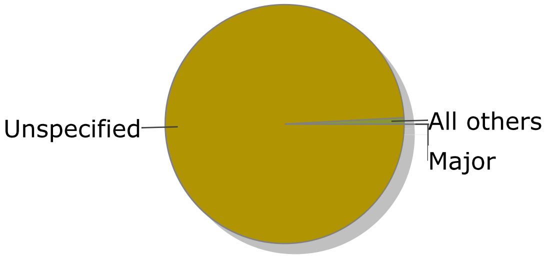
Medium-impact defects that cause incorrect results, concurrency problems, and system freezes.



Defect Risk by Component

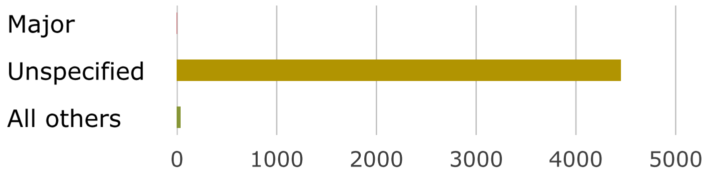
Component	Owner	Defect Density
Drivers		0.50
Drivers-Net		0.52
Other		0.91
Drivers-Staging		0.90
FS		0.73
Networking		0.62
Sound		0.32
Kernel		0.73
Drivers-ISDN		0.49
KVM		1.54
All Others		0.43





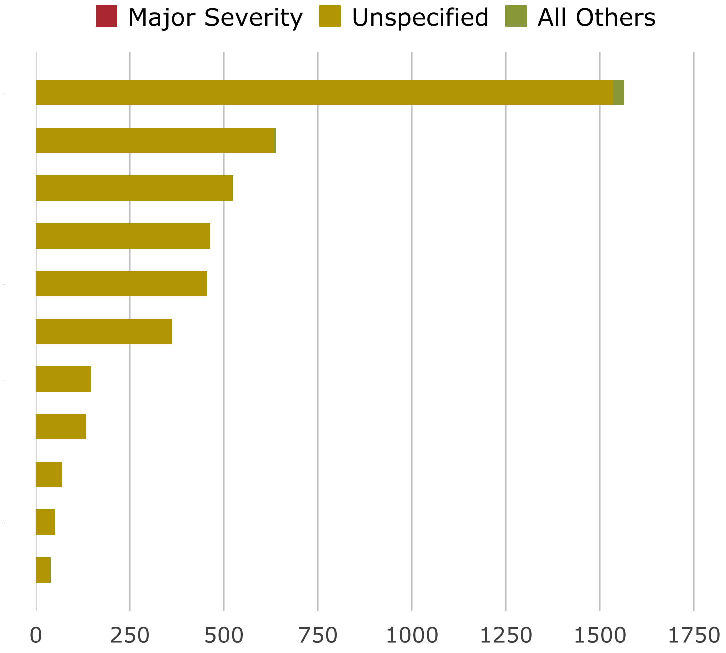
Defects by Assigned Severity

High-severity defects have been tagged by developers as a clear threat to the program's stability and/or security.

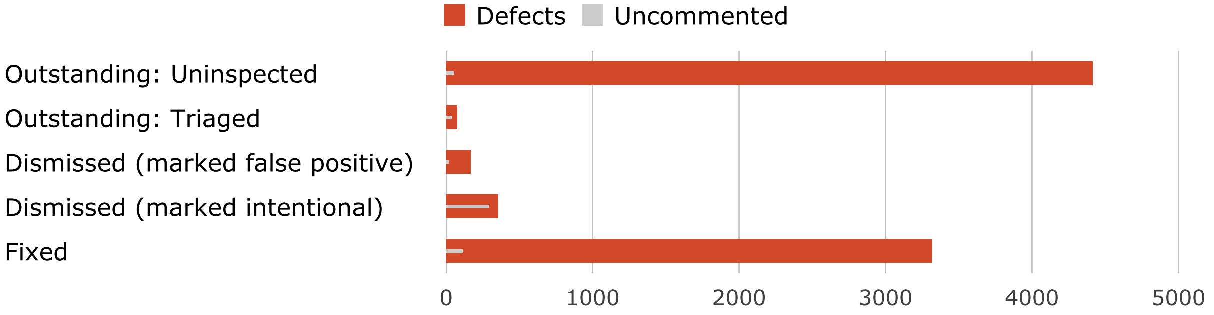


Defect Severities by Component

Component	Owner	Defect Density
Drivers		0.50
Drivers-Net		0.52
Other		0.91
Drivers-Staging		0.90
FS		0.73
Networking		0.62
Sound		0.32
Kernel		0.73
Drivers-ISDN		0.49
KVM		1.54
All Others		0.43



Defects by Triage State



Coverity Software Integrity Report

The Coverity Software Integrity Rating is an objective standard used by developers, management, and business executives to assess the software integrity level of the code they are shipping in their products and systems.

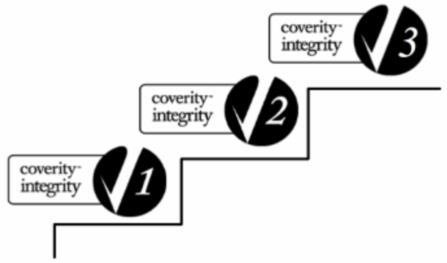
Coverity rating requirements are based on an assessment of several factors:

- Defect density: For a given component or code base, the number of high-risk and medium-risk defects found by static analysis divided by the lines of code analyzed. Defect density excludes fixed defects and defects dismissed as false positives or intentional. For example, if there are 100 high-risk and medium-risk defects found by static analysis in a code base of 100,000 lines of code, the defect density would be 100/100,000 = 1 defect per thousand lines of code.
- Major severity defects: Developers can assess the severity of defects by marking them as Major, Moderate, or Minor (customizations might affect these labels). We consider all defects assigned a severity rating of Major to be worth reporting in the Integrity Report regardless of their risk level because the severity rating is manually assigned by a developer who has reviewed the defect.
- False positive rate: Developers can mark defect reports as false positives if they are not real defects. We consider a false positive rate of less than 20% to be normal for Coverity Static Analysis. A false positive rate above 20% indicates possible misconfiguration, incorrect inspection, use of unusual idioms in the code, or a flaw in our analysis.

Coverity Integrity Level 1 requires the software has less than or equal to 1 defect per thousand lines of code, which is approximately the average defect density for the software industry.

Coverity Integrity Level 2 requires the software has less than or equal to 0.1 defect per thousand lines of code, which is approximately at the 90th percentile for the software industry. This is a much higher bar to satisfy than Level 1. A million-line code base would have to have 100 or fewer defects to qualify for Level 2.

Coverity Integrity Level 3 This is the highest bar in the rating system today. All three of the following criteria need to be met:



- Defect density less than or equal to 0.01 defect per thousand lines of code, which is approximately in the 99th percentile for the software industry. This means that a million-line code base must have ten or fewer defects remaining. The requirement does not specify zero defects because this might force the delay of a release for a few stray static analysis defects that are not in a critical component (or else giving up on achieving a target Level 3 for the release).
- False positive rate less than 20%. If the rate is higher the results need to be audited by Coverity to qualify for this integrity rating level. A higher false positive rate indicates misconfiguration, usage of unusual idioms, or incorrect diagnosis of a large number of defects. The Coverity Static Analysis has less than 20% false positives for most code bases, so we reserve the right to audit false positives when they exceed this threshold.
- Zero defects marked as Major severity by the user. The severity of each defect can be set to Major, Moderate, or Minor. This requirement ensures that all defects marked as Major by the user are fixed because we believe that once human judgment has been applied, all Major defects must be fixed to achieve Level 3.

Level Not Achieved indicates that the target level criteria are not met. This means that the software has too many unresolved static analysis defects in it to qualify for the desired target integrity level. To achieve the target integrity level rating, more defects should be reviewed and fixed.

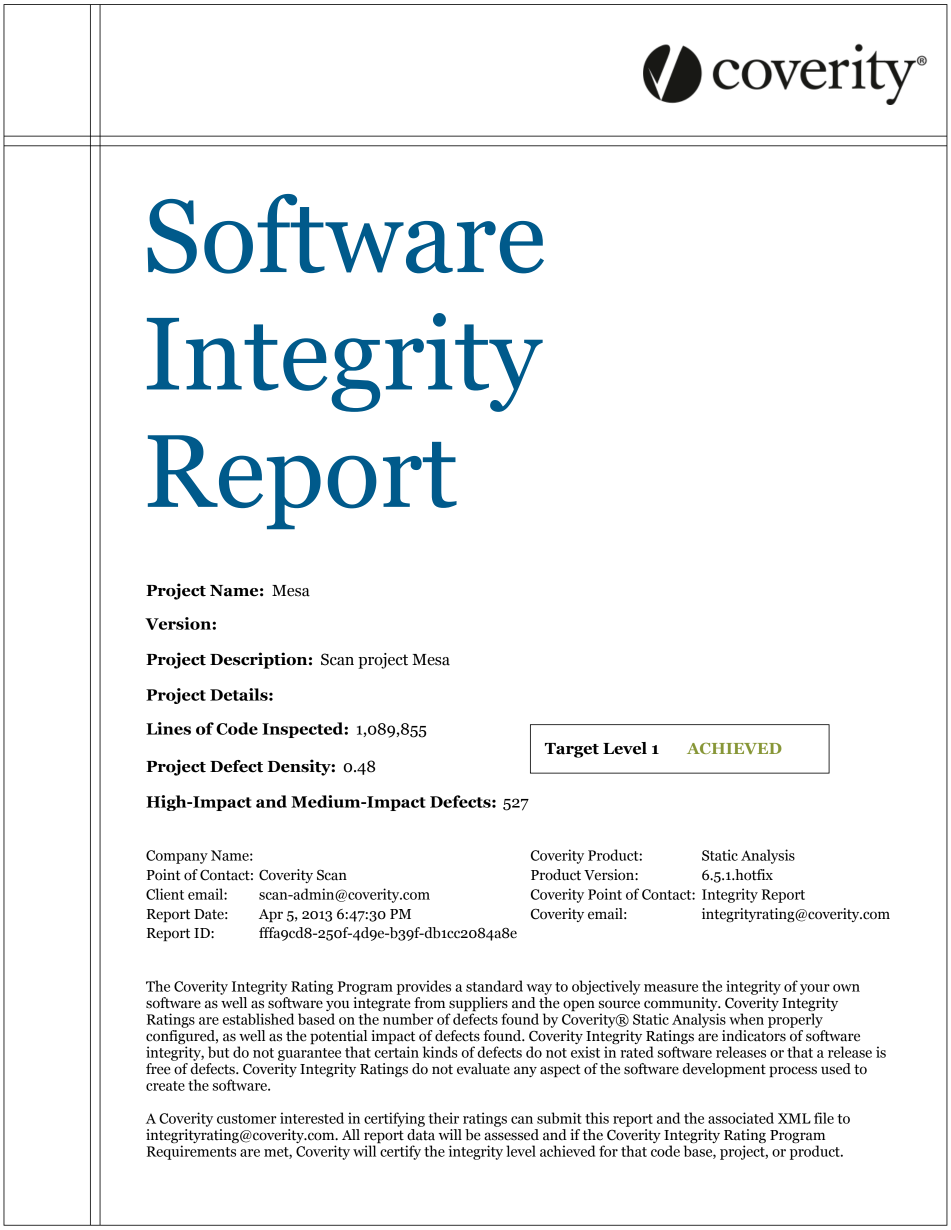
How to Use Your Software Integrity Rating

Set software integrity standards for your projects, products, and teams.
It is often difficult for developers and development management to objectively compare the integrity of code bases, projects, and products. The Coverity Software Integrity Rating is a way to create "apples-to-apples" comparisons and promote the success of development teams that consistently deliver highly-rated software code and products. Development teams can also use these ratings as objective evidence to satisfy requirements for quality and safety standards.

Audit your software supply chain.
It is challenging for companies to assess the integrity of software code from suppliers and partners that they integrate with their offerings. The Coverity Software Integrity Rating is a way to help companies create a common measurement of software integrity across their entire software supply chain.

Promote your commitment to software integrity.
The integrity of your software has a direct impact on the integrity of your brand. Showcasing your commitment to software integrity is a valuable way to boost your brand value. It indicates that they are committed to delivering software that is safe, secure, and performs as expected.

Appendix D: Coverity Software Integrity Report for Mesa



coverity®

Software Integrity Report

Project Name: Mesa

Version:

Project Description: Scan project Mesa

Project Details:

Lines of Code Inspected: 1,089,855

Project Defect Density: 0.48

High-Impact and Medium-Impact Defects: 527

Target Level 1 ACHIEVED

Company Name:		Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 5, 2013 6:47:30 PM	Coverity email:	integrityrating@coverity.com
Report ID:	fffa9cd8-250f-4d9e-b39f-db1cc2084a8e		

The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.

coverity®

Software Integrity Report

Project Name: Mesa

Version:

Project Description: Scan project Mesa

Project Details:

Lines of Code Inspected: 1,089,855

Project Defect Density: 0.48

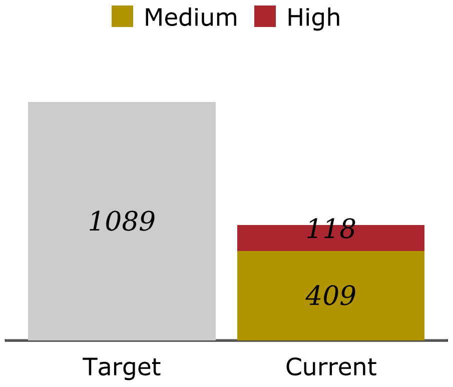
High-Impact and Medium-Impact Defects: 527

Target Level 1 ACHIEVED

Company Name:		Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 5, 2013 6:47:30 PM	Coverity email:	integrityrating@coverity.com
Report ID:	fffa9cd8-250f-4d9e-b39f-db1cc2084a8e		

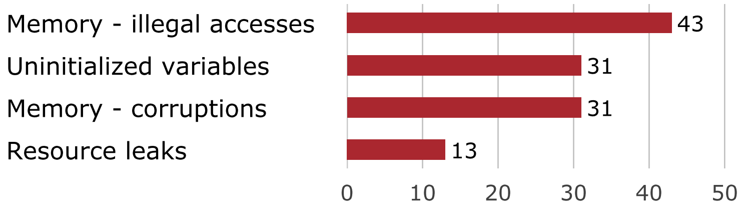
The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.



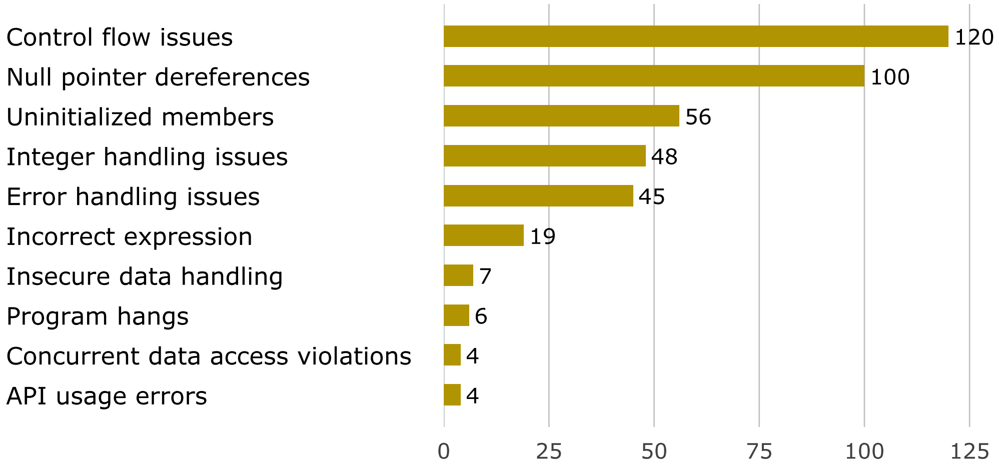
High-Risk Defects

High-impact defects that cause crashes, program instability, and performance problems.



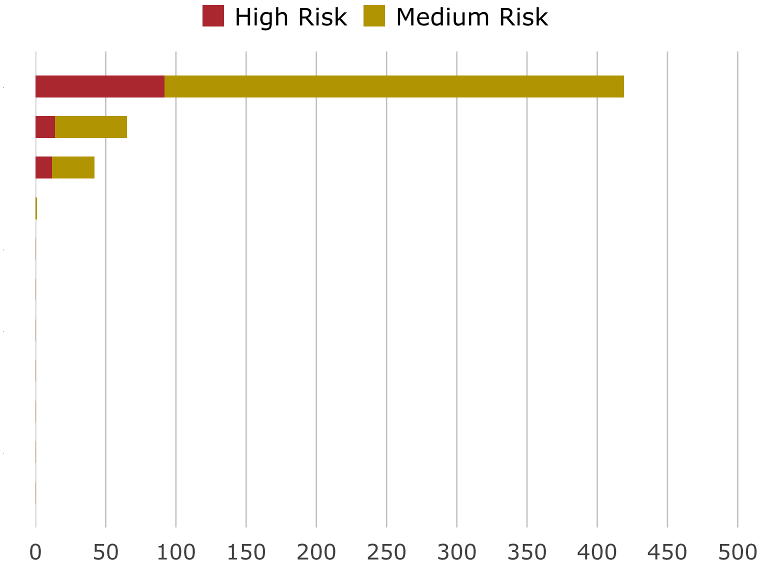
Medium-Risk Defects

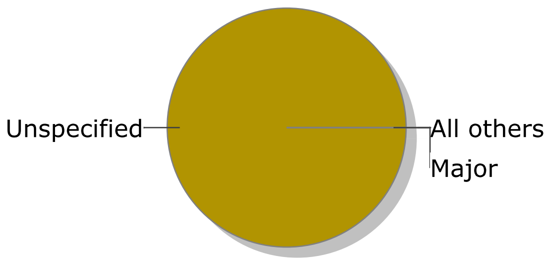
Medium-impact defects that cause incorrect results, concurrency problems, and system freezes.



Defect Risk by Component

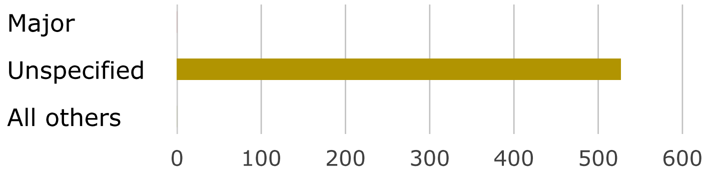
Component	Owner	Defect Density
Other		0.51
Drivers		0.50
Main		0.31
Libdricore		0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00





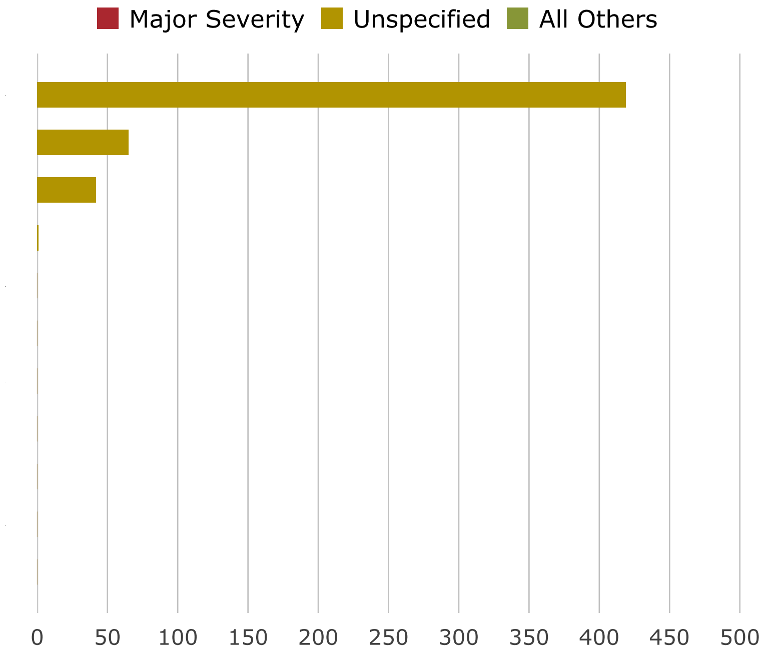
Defects by Assigned Severity

High-severity defects have been tagged by developers as a clear threat to the program's stability and/or security.

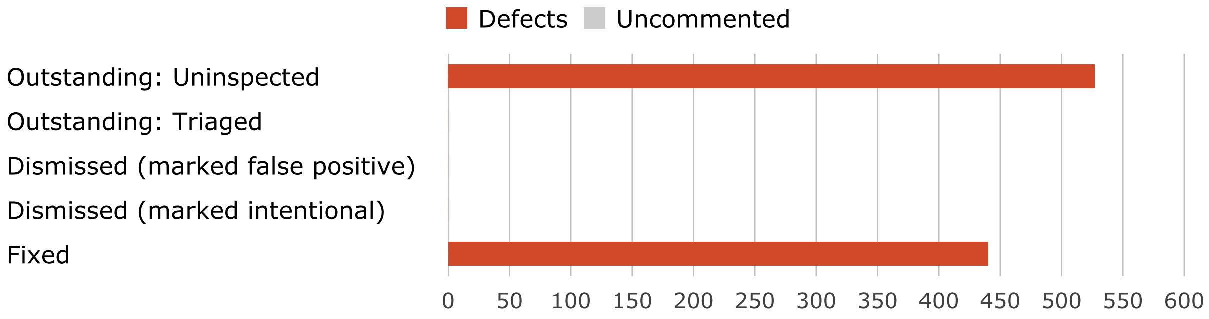


Defect Severities by Component

Component	Owner	Defect Density
Other		0.51
Drivers		0.50
Main		0.31
Libdricore		0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00



Defects by Triage State



Coverity Software Integrity Report

The Coverity Software Integrity Rating is an objective standard used by developers, management, and business executives to assess the software integrity level of the code they are shipping in their products and systems.

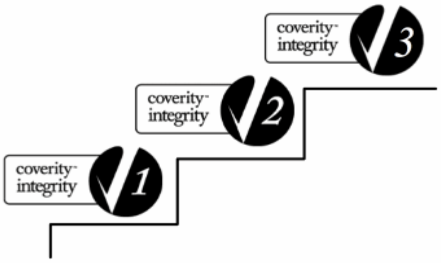
Coverity rating requirements are based on an assessment of several factors:

- Defect density: For a given component or code base, the number of high-risk and medium-risk defects found by static analysis divided by the lines of code analyzed. Defect density excludes fixed defects and defects dismissed as false positives or intentional. For example, if there are 100 high-risk and medium-risk defects found by static analysis in a code base of 100,000 lines of code, the defect density would be 100/100,000 = 1 defect per thousand lines of code.
- Major severity defects: Developers can assess the severity of defects by marking them as Major, Moderate, or Minor (customizations might affect these labels). We consider all defects assigned a severity rating of Major to be worth reporting in the Integrity Report regardless of their risk level because the severity rating is manually assigned by a developer who has reviewed the defect.
- False positive rate: Developers can mark defect reports as false positives if they are not real defects. We consider a false positive rate of less than 20% to be normal for Coverity Static Analysis. A false positive rate above 20% indicates possible misconfiguration, incorrect inspection, use of unusual idioms in the code, or a flaw in our analysis.

Coverity Integrity Level 1 requires the software has less than or equal to 1 defect per thousand lines of code, which is approximately the average defect density for the software industry.

Coverity Integrity Level 2 requires the software has less than or equal to 0.1 defect per thousand lines of code, which is approximately at the 90th percentile for the software industry. This is a much higher bar to satisfy than Level 1. A million-line code base would have to have 100 or fewer defects to qualify for Level 2.

Coverity Integrity Level 3 This is the highest bar in the rating system today. All three of the following criteria need to be met:



- Defect density less than or equal to 0.01 defect per thousand lines of code, which is approximately in the 99th percentile for the software industry. This means that a million-line code base must have ten or fewer defects remaining. The requirement does not specify zero defects because this might force the delay of a release for a few stray static analysis defects that are not in a critical component (or else giving up on achieving a target Level 3 for the release).
- False positive rate less than 20%. If the rate is higher the results need to be audited by Coverity to qualify for this integrity rating level. A higher false positive rate indicates misconfiguration, usage of unusual idioms, or incorrect diagnosis of a large number of defects. The Coverity Static Analysis has less than 20% false positives for most code bases, so we reserve the right to audit false positives when they exceed this threshold.
- Zero defects marked as Major severity by the user. The severity of each defect can be set to Major, Moderate, or Minor. This requirement ensures that all defects marked as Major by the user are fixed because we believe that once human judgment has been applied, all Major defects must be fixed to achieve Level 3.

Level Not Achieved indicates that the target level criteria are not met. This means that the software has too many unresolved static analysis defects in it to qualify for the desired target integrity level. To achieve the target integrity level rating, more defects should be reviewed and fixed.

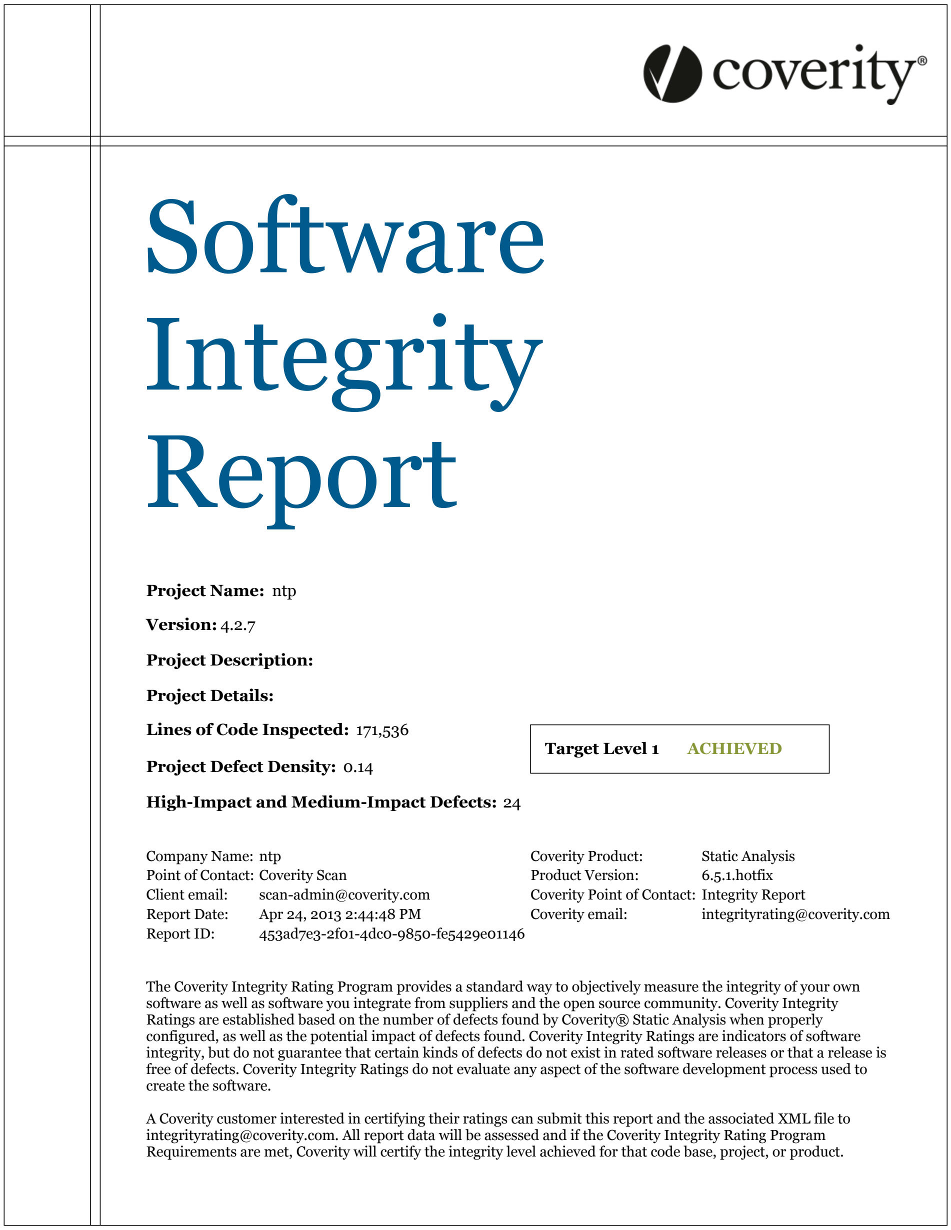
How to Use Your Software Integrity Rating

Set software integrity standards for your projects, products, and teams.
It is often difficult for developers and development management to objectively compare the integrity of code bases, projects, and products. The Coverity Software Integrity Rating is a way to create "apples-to-apples" comparisons and promote the success of development teams that consistently deliver highly-rated software code and products. Development teams can also use these ratings as objective evidence to satisfy requirements for quality and safety standards.

Audit your software supply chain.
It is challenging for companies to assess the integrity of software code from suppliers and partners that they integrate with their offerings. The Coverity Software Integrity Rating is a way to help companies create a common measurement of software integrity across their entire software supply chain.

Promote your commitment to software integrity.
The integrity of your software has a direct impact on the integrity of your brand. Showcasing your commitment to software integrity is a valuable way to boost your brand value. It indicates that they are committed to delivering software that is safe, secure, and performs as expected.

Appendix E: Coverity Software Integrity Report for NTP



coverity®

Software Integrity Report

Project Name: ntp

Version: 4.2.7

Project Description:

Project Details:

Lines of Code Inspected: 171,536

Project Defect Density: 0.14

High-Impact and Medium-Impact Defects: 24

Target Level 1 ACHIEVED

Company Name:	ntp	Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 24, 2013 2:44:48 PM	Coverity email:	integrityrating@coverity.com
Report ID:	453ad7e3-2f01-4dc0-9850-fe5429e01146		

The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.

Software Integrity Report

Project Name: ntp

Version: 4.2.7

Project Description:

Project Details:

Lines of Code Inspected: 171,536

Project Defect Density: 0.14

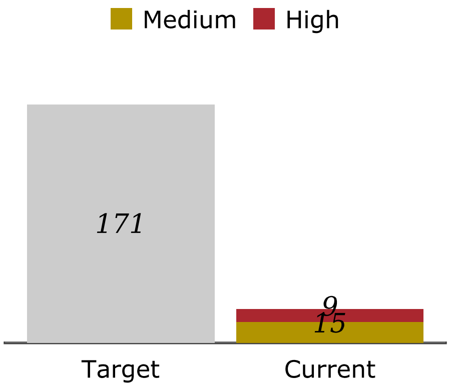
High-Impact and Medium-Impact Defects: 24

Target Level 1 ACHIEVED

Company Name:	ntp	Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 24, 2013 2:44:48 PM	Coverity email:	integrityrating@coverity.com
Report ID:	453ad7e3-2f01-4dc0-9850-fe5429e01146		

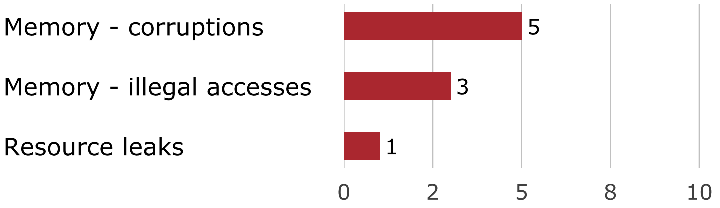
The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.



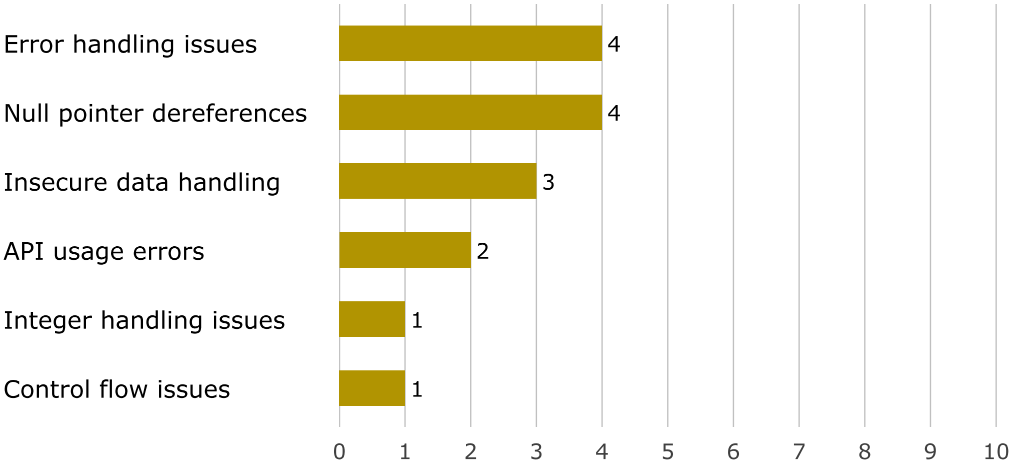
High-Risk Defects

High-impact defects that cause crashes, program instability, and performance problems.



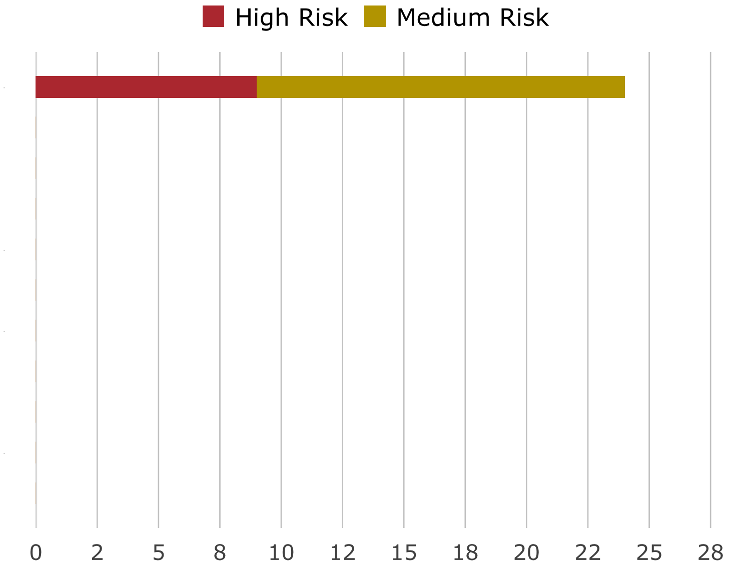
Medium-Risk Defects

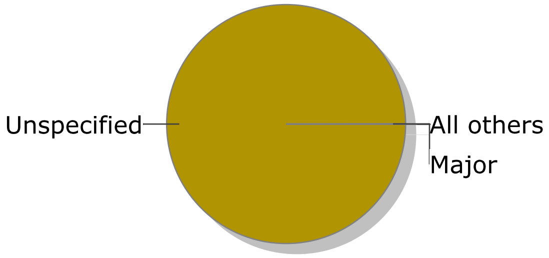
Medium-impact defects that cause incorrect results, concurrency problems, and system freezes.



Defect Risk by Component

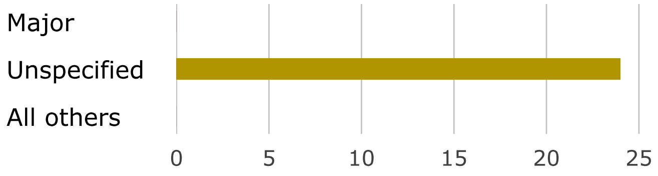
Component	Owner	Defect Density
Other		0.14
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00





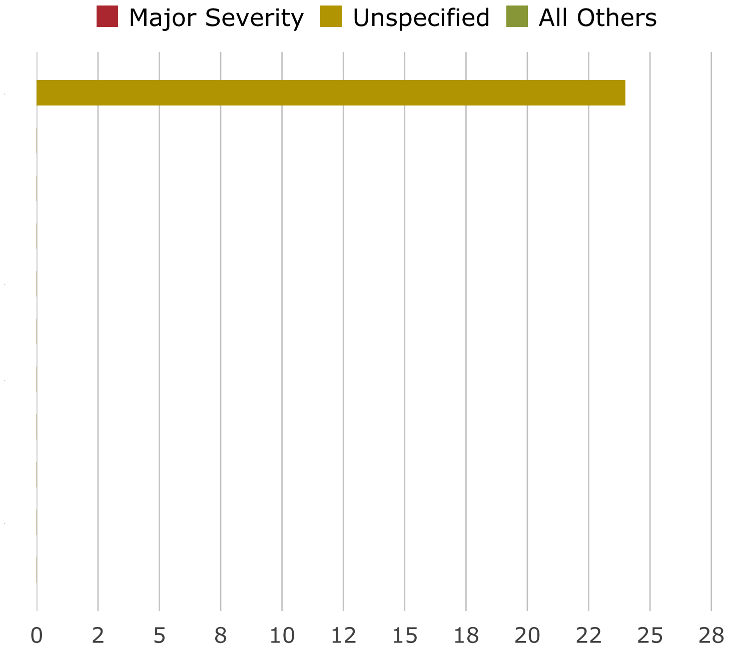
Defects by Assigned Severity

High-severity defects have been tagged by developers as a clear threat to the program's stability and/or security.

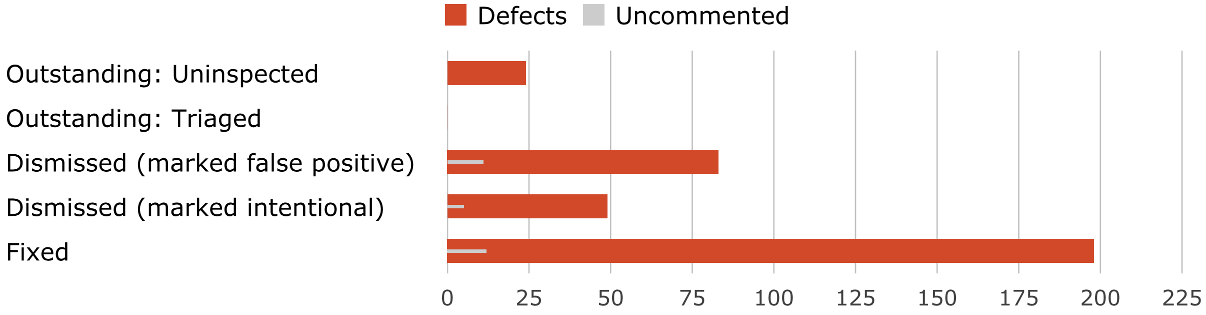


Defect Severities by Component

Component	Owner	Defect Density
Other		0.14
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00
-	-	0.00



Defects by Triage State



Coverity Software Integrity Report

The Coverity Software Integrity Rating is an objective standard used by developers, management, and business executives to assess the software integrity level of the code they are shipping in their products and systems.

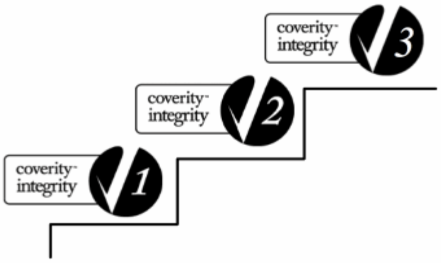
Coverity rating requirements are based on an assessment of several factors:

- Defect density: For a given component or code base, the number of high-risk and medium-risk defects found by static analysis divided by the lines of code analyzed. Defect density excludes fixed defects and defects dismissed as false positives or intentional. For example, if there are 100 high-risk and medium-risk defects found by static analysis in a code base of 100,000 lines of code, the defect density would be 100/100,000 = 1 defect per thousand lines of code.
- Major severity defects: Developers can assess the severity of defects by marking them as Major, Moderate, or Minor (customizations might affect these labels). We consider all defects assigned a severity rating of Major to be worth reporting in the Integrity Report regardless of their risk level because the severity rating is manually assigned by a developer who has reviewed the defect.
- False positive rate: Developers can mark defect reports as false positives if they are not real defects. We consider a false positive rate of less than 20% to be normal for Coverity Static Analysis. A false positive rate above 20% indicates possible misconfiguration, incorrect inspection, use of unusual idioms in the code, or a flaw in our analysis.

Coverity Integrity Level 1 requires the software has less than or equal to 1 defect per thousand lines of code, which is approximately the average defect density for the software industry.

Coverity Integrity Level 2 requires the software has less than or equal to 0.1 defect per thousand lines of code, which is approximately at the 90th percentile for the software industry. This is a much higher bar to satisfy than Level 1. A million-line code base would have to have 100 or fewer defects to qualify for Level 2.

Coverity Integrity Level 3 This is the highest bar in the rating system today. All three of the following criteria need to be met:



- Defect density less than or equal to 0.01 defect per thousand lines of code, which is approximately in the 99th percentile for the software industry. This means that a million-line code base must have ten or fewer defects remaining. The requirement does not specify zero defects because this might force the delay of a release for a few stray static analysis defects that are not in a critical component (or else giving up on achieving a target Level 3 for the release).
- False positive rate less than 20%. If the rate is higher the results need to be audited by Coverity to qualify for this integrity rating level. A higher false positive rate indicates misconfiguration, usage of unusual idioms, or incorrect diagnosis of a large number of defects. The Coverity Static Analysis has less than 20% false positives for most code bases, so we reserve the right to audit false positives when they exceed this threshold.
- Zero defects marked as Major severity by the user. The severity of each defect can be set to Major, Moderate, or Minor. This requirement ensures that all defects marked as Major by the user are fixed because we believe that once human judgment has been applied, all Major defects must be fixed to achieve Level 3.

Level Not Achieved indicates that the target level criteria are not met. This means that the software has too many unresolved static analysis defects in it to qualify for the desired target integrity level. To achieve the target integrity level rating, more defects should be reviewed and fixed.

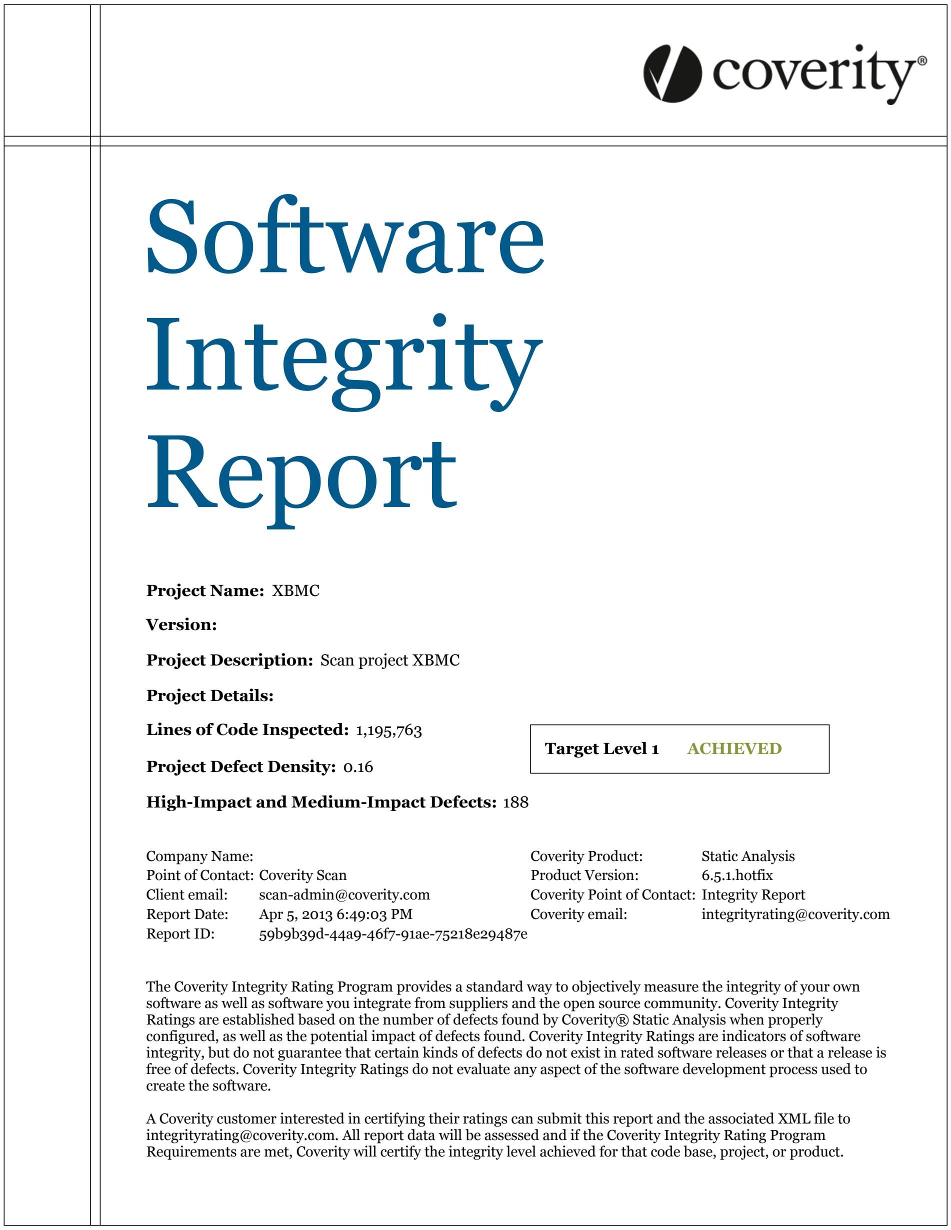
How to Use Your Software Integrity Rating

Set software integrity standards for your projects, products, and teams.
It is often difficult for developers and development management to objectively compare the integrity of code bases, projects, and products. The Coverity Software Integrity Rating is a way to create "apples-to-apples" comparisons and promote the success of development teams that consistently deliver highly-rated software code and products. Development teams can also use these ratings as objective evidence to satisfy requirements for quality and safety standards.

Audit your software supply chain.
It is challenging for companies to assess the integrity of software code from suppliers and partners that they integrate with their offerings. The Coverity Software Integrity Rating is a way to help companies create a common measurement of software integrity across their entire software supply chain.

Promote your commitment to software integrity.
The integrity of your software has a direct impact on the integrity of your brand. Showcasing your commitment to software integrity is a valuable way to boost your brand value. It indicates that they are committed to delivering software that is safe, secure, and performs as expected.

Appendix F: Coverity Software Integrity Report for XBMC



coverity®

Software Integrity Report

Project Name: XBMC

Version:

Project Description: Scan project XBMC

Project Details:

Lines of Code Inspected: 1,195,763

Project Defect Density: 0.16

High-Impact and Medium-Impact Defects: 188

Target Level 1 ACHIEVED

Company Name:		Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 5, 2013 6:49:03 PM	Coverity email:	integrityrating@coverity.com
Report ID:	59b9b39d-44a9-46f7-91ae-75218e29487e		

The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.

Software Integrity Report

Project Name: XBMC

Version:

Project Description: Scan project XBMC

Project Details:

Lines of Code Inspected: 1,195,763

Project Defect Density: 0.16

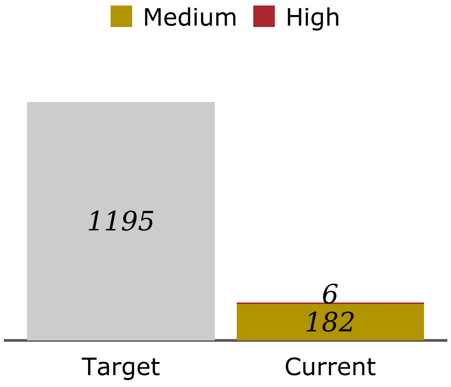
High-Impact and Medium-Impact Defects: 188

Target Level 1 ACHIEVED

Company Name:		Coverity Product:	Static Analysis
Point of Contact:	Coverity Scan	Product Version:	6.5.1.hotfix
Client email:	scan-admin@coverity.com	Coverity Point of Contact:	Integrity Report
Report Date:	Apr 5, 2013 6:49:03 PM	Coverity email:	integrityrating@coverity.com
Report ID:	59b9b39d-44a9-46f7-91ae-75218e29487e		

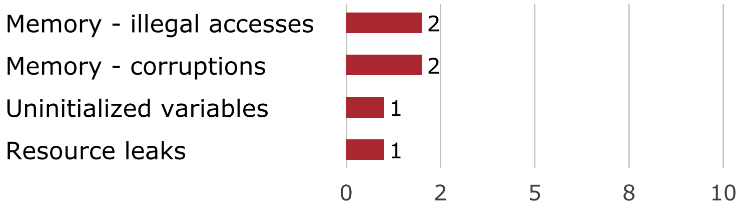
The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis when properly configured, as well as the potential impact of defects found. Coverity Integrity Ratings are indicators of software integrity, but do not guarantee that certain kinds of defects do not exist in rated software releases or that a release is free of defects. Coverity Integrity Ratings do not evaluate any aspect of the software development process used to create the software.

A Coverity customer interested in certifying their ratings can submit this report and the associated XML file to integrityrating@coverity.com. All report data will be assessed and if the Coverity Integrity Rating Program Requirements are met, Coverity will certify the integrity level achieved for that code base, project, or product.



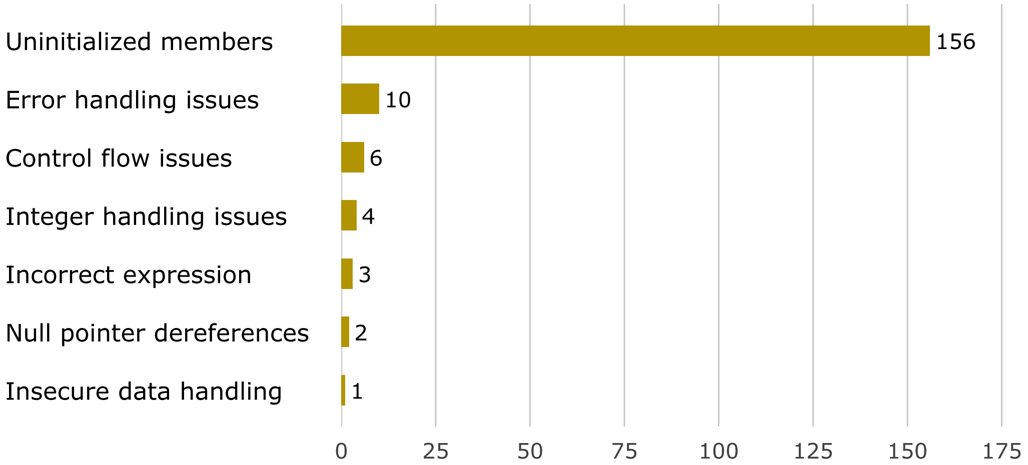
High-Risk Defects

High-impact defects that cause crashes, program instability, and performance problems.



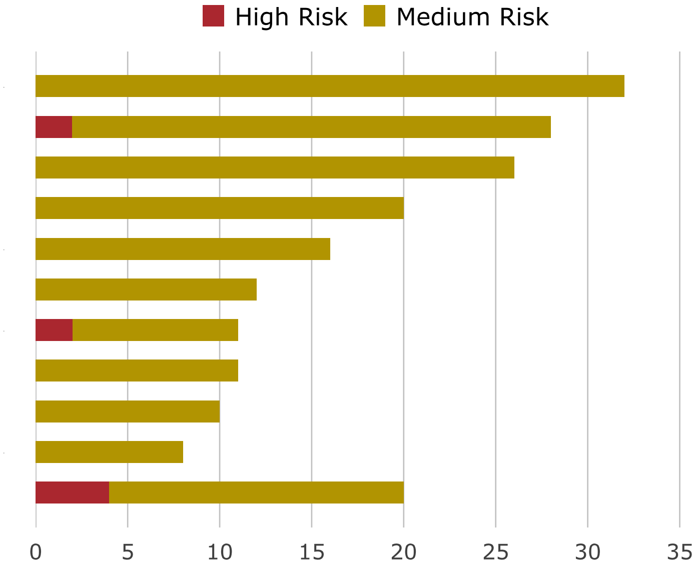
Medium-Risk Defects

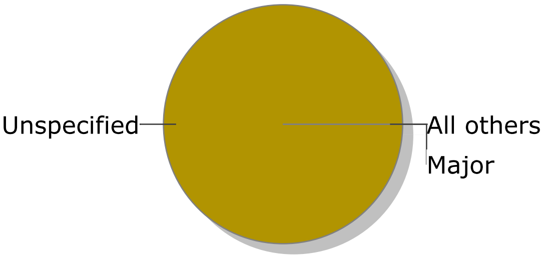
Medium-impact defects that cause incorrect results, concurrency problems, and system freezes.



Defect Risk by Component

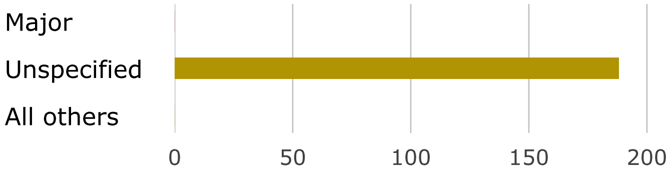
Component	Owner	Defect Density
Other		0.03
filesystem		0.95
core		0.46
music		1.09
interfaces		0.40
network		0.90
video		0.44
addons		0.98
dialogs		1.28
pvr		0.51
All Others		0.29





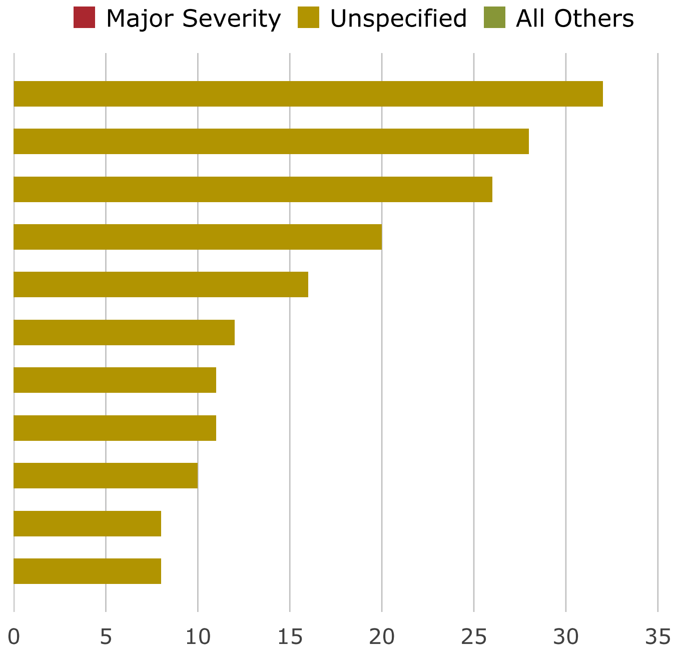
Defects by Assigned Severity

High-severity defects have been tagged by developers as a clear threat to the program's stability and/or security.

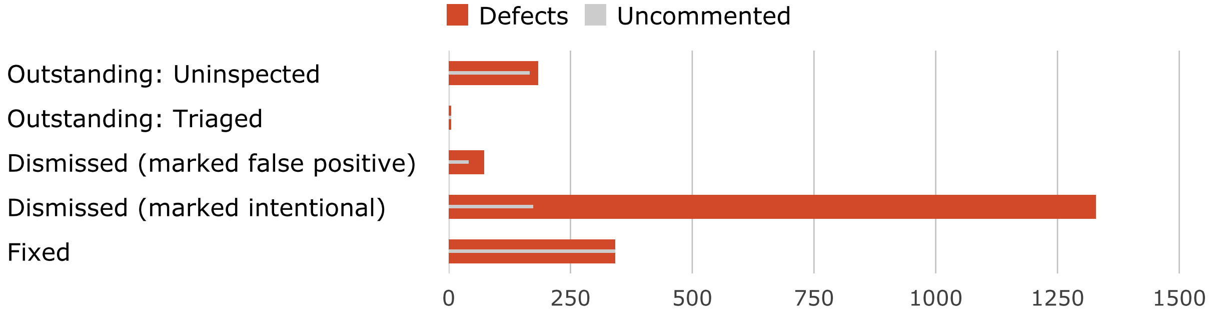


Defect Severities by Component

Component	Owner	Defect Density
Other		0.03
filesystem		0.95
core		0.46
music		1.09
interfaces		0.40
network		0.90
video		0.44
addons		0.98
dialogs		1.28
pvr		0.51
All Others		0.29



Defects by Triage State



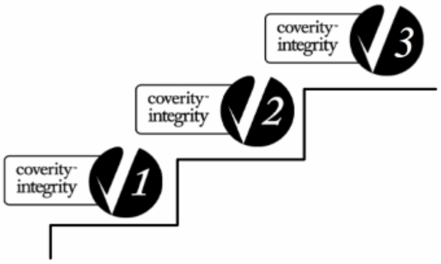
Coverity Software Integrity Report

The Coverity Software Integrity Rating is an objective standard used by developers, management, and business executives to assess the software integrity level of the code they are shipping in their products and systems.

Coverity rating requirements are based on an assessment of several factors:

- Defect density: For a given component or code base, the number of high-risk and medium-risk defects found by static analysis divided by the lines of code analyzed. Defect density excludes fixed defects and defects dismissed as false positives or intentional. For example, if there are 100 high-risk and medium-risk defects found by static analysis in a code base of 100,000 lines of code, the defect density would be 100/100,000 = 1 defect per thousand lines of code.
- Major severity defects: Developers can assess the severity of defects by marking them as Major, Moderate, or Minor (customizations might affect these labels). We consider all defects assigned a severity rating of Major to be worth reporting in the Integrity Report regardless of their risk level because the severity rating is manually assigned by a developer who has reviewed the defect.
- False positive rate: Developers can mark defect reports as false positives if they are not real defects. We consider a false positive rate of less than 20% to be normal for Coverity Static Analysis. A false positive rate above 20% indicates possible misconfiguration, incorrect inspection, use of unusual idioms in the code, or a flaw in our analysis.

Coverity Integrity Level 1 requires the software has less than or equal to 1 defect per thousand lines of code, which is approximately the average defect density for the software industry.



Coverity Integrity Level 2 requires the software has less than or equal to 0.1 defect per thousand lines of code, which is approximately at the 90th percentile for the software industry. This is a much higher bar to satisfy than Level 1. A million-line code base would have to have 100 or fewer defects to qualify for Level 2.

Coverity Integrity Level 3 This is the highest bar in the rating system today. All three of the following criteria need to be met:

- Defect density less than or equal to 0.01 defect per thousand lines of code, which is approximately in the 99th percentile for the software industry. This means that a million-line code base must have ten or fewer defects remaining. The requirement does not specify zero defects because this might force the delay of a release for a few stray static analysis defects that are not in a critical component (or else giving up on achieving a target Level 3 for the release).
- False positive rate less than 20%. If the rate is higher the results need to be audited by Coverity to qualify for this integrity rating level. A higher false positive rate indicates misconfiguration, usage of unusual idioms, or incorrect diagnosis of a large number of defects. The Coverity Static Analysis has less than 20% false positives for most code bases, so we reserve the right to audit false positives when they exceed this threshold.
- Zero defects marked as Major severity by the user. The severity of each defect can be set to Major, Moderate, or Minor. This requirement ensures that all defects marked as Major by the user are fixed because we believe that once human judgment has been applied, all Major defects must be fixed to achieve Level 3.

Level Not Achieved indicates that the target level criteria are not met. This means that the software has too many unresolved static analysis defects in it to qualify for the desired target integrity level. To achieve the target integrity level rating, more defects should be reviewed and fixed.

How to Use Your Software Integrity Rating

Set software integrity standards for your projects, products, and teams.
It is often difficult for developers and development management to objectively compare the integrity of code bases, projects, and products. The Coverity Software Integrity Rating is a way to create "apples-to-apples" comparisons and promote the success of development teams that consistently deliver highly-rated software code and products. Development teams can also use these ratings as objective evidence to satisfy requirements for quality and safety standards.

Audit your software supply chain.
It is challenging for companies to assess the integrity of software code from suppliers and partners that they integrate with their offerings. The Coverity Software Integrity Rating is a way to help companies create a common measurement of software integrity across their entire software supply chain.

Promote your commitment to software integrity.
The integrity of your software has a direct impact on the integrity of your brand. Showcasing your commitment to software integrity is a valuable way to boost your brand value. It indicates that they are committed to delivering software that is safe, secure, and performs as expected.