

2017 Coverity Scan Report

Open Source Software—The Road Ahead

Mel Llaguno, Open Source Solution Manager

TABLE OF CONTENTS

Executive summary	3
A brief history	3
Scan today	4
What have we learned since we started?.....	5
OSS maturity.....	6
Scan’s community.....	10
Improvements to Scan.....	11
Case study: Quality, security, and project maturity	13
The future	16
From defect density to maturity.....	16
The Census Project.....	17
Community Health Analytics Open Source Software	18
A new view of maturity.....	18
Conclusion	19

Executive summary

Coverity Scan's impact on open source software (OSS) is both extensive and largely unacknowledged. This report discusses various aspects of the community and projects related to Scan and highlights both the contribution Scan has made to the maturity of the development practices of OSS projects and the impact it has had on the quality and security of the OSS ecosystem. In addition, we examine historical perspectives regarding the use of defect density as the sole measure of quality. We then expand on the perspective required to measure OSS project risk and examine initiatives from the Linux Foundation that may potentially be incorporated into Scan to provide a holistic view of a project.

A brief history

Coverity Scan started in 2006 as one of the largest public-private sector research projects initiated with the U.S. Department of Homeland Security (DHS) with a focus on OSS integrity. The affiliation with DHS concluded in 2009, but today, over a decade after its inception, Coverity Scan continues to provide best-in-class static analysis tools for OSS projects.

Back in the early days of Scan's inception, the use of OSS was still on the periphery of commercial software development. Microsoft was still largely dominant in all aspects of computing, and while a fervent few believed in the future of OSS, it was the growing popularity of Linux within enterprise environments that made people question the suitability of OSS for commercial applications. Looking back at the types of questions we've asked in our previous Scan reports, we see a progression toward the maturity and general acceptance of OSS, but also increasing awareness of our collective risk.

2008:

What kinds of defects are you finding in OSS? Is it safe? Should I use it?

2009:

Is OSS getting better or worse? Are defects changing? Are developers still fixing them?

2010:

Can I get defect visibility in the OSS I'm using? Can you tell me what I'm shipping?

2011:

How does the quality of OSS projects compare with their commercial counterparts?

2012:

Are defect densities in OSS projects showing an improvement or a degradation?

2013:

How does the quality of C/C++ projects compare to Java?

2014:

What does OSS quality and security look like in the era of Heartbleed?

* Note: Coverity Scan reports are published for the previous year (meaning the 2014 report was published in 2015).

In addition to providing a comprehensive analysis on the evolving state of Linux, we continued to expand visibility to other OSS projects, such as Android, PostgreSQL, and PHP.

More interestingly, consider the past predictions supplied by Gartner:

- In our 2010 Scan report, Gartner estimated that "by 2012, at least 80% of commercial software packages would include elements of OSS technology."
- In our 2011 Scan report, Gartner estimated that by "2016, OSS will be included in mission-critical software portfolios within 99% of Global 2000 enterprises."

A brief history (continued)

Today, OSS development is one of the primary driving forces of technological innovation, and previous detractors, such as Microsoft, have wholeheartedly embraced OSS. From artificial intelligence to the Internet of Things, autonomous driving, distributed ledgers, and cloud computing infrastructure, OSS plays a pivotal role in the evolution of a wide range of technologies.

OSS has won. But what are the implications of its dominance?

To answer this question, we need to look back to 2015 to provide context for the current state of affairs. Heartbleed was arguably one of the most visible security vulnerabilities in the modern era. Abstracting the technical details into an instantly recognizable moniker allowed the general public to participate in the discussion regarding information security. This awareness revealed the widespread interdependence of the modern OSS ecosystem. No longer were security issues associated with OSS relegated to highly technical forums; instead, the public at large became aware of these issues due to the pervasive impact on their lives. The success of OSS can be measured by the scale of its impact on general users of technology and our exposure to the collective risks of consuming OSS. Now mainstream, OSS is succumbing to the pressure to mature, given the critical role it plays in modern technology.

Given Scan's historical role in helping mature the OSS ecosystem up to this point, we will examine future directions where Scan can help promote software maturity within the OSS community.

Scan today

What have we learned since we started?

In previous Scan reports, we made a number of bold assertions:

“Open source quality for active projects in Coverity Scan is better than the software industry average.”

“Open source code quality surpasses proprietary code quality in C/C++ projects.”

In total, Scan has identified **over 1.1 million defects** in active OSS projects, with **over 600,000 fixed defects.**

Today, the distinction between proprietary/commercial and open source is irrelevant. According to some of the largest commercial users of Coverity, software being shipped to customers can contain up to 90% open source code. In addition, there are now companies founded entirely on OSS. OSS is now the norm.

While the success of OSS is unquestionable, its adoption highlights a growing concern regarding the quality, security, and maturity of the software itself. Many people attribute the quality of OSS to Linus' law, which states, “Given enough eyeballs, all bugs are shallow.” Unfortunately, simply making code publicly accessible for review is no guarantee of quality, as we found out the hard way with Heartbleed. Gaps in quality inevitably compromise security.

“The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This

Scan today

(continued)

compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.”

—<http://heartbleed.com>

Even a tool like Scan was initially unable to identify the defect until we understood the correct heuristic to identify this class of vulnerability.

You can read about how we changed our analysis to detect Heartbleed in this blog post by one of Coverity's founders, Andy Chou:

<https://www.synopsys.com/blogs/software-security/detecting-heartbleed-with-static-analysis/>

This presents a challenge for adopters/consumers of OSS—how to determine the fitness of a particular OSS project.

While static analysis has been extremely beneficial for improving the quality and security of OSS, other software integrity techniques (such as software fuzzing, used to verify the existence of Heartbleed) in combination with broader project health/community metrics may be necessary to develop a complete picture of maturity.

“Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The program is then monitored for exceptions such as crashes, or failing built-in code assertions or for finding potential memory leaks. Typically, fuzzers are used to test programs that take structured inputs. This structure is specified, e.g., in a file format or protocol and distinguishes valid from invalid input. An effective fuzzer generates semi-valid inputs that are ‘valid enough’ in that they are not directly rejected by the parser, but do create unexpected behaviors deeper in the program and are ‘invalid enough’ to expose corner cases that have not been properly dealt with.

For the purpose of security, input that crosses a trust boundary is often the most interesting. For example, it is more important to fuzz code that handles the upload of a file by any user than it is to fuzz the code that parses a configuration file that is accessible only to a privileged user.”

—Wikipedia

Scan today

(continued)

Total projects since Scan's inception

In the past, we did not disclose how we determined an active project within Scan. Going forward, our criteria will require projects that are approved to submit more than a single, initial build. This ensures that at the very minimum, results between analysis runs are being compared.

For those who are unfamiliar with the submission criteria for Scan, we require a contributor/owner of the OSS project to submit a publicly accessible source code repository for verification of the OSS license. Only contributors can submit projects to Scan. This ensures that any potentially sensitive issues uncovered by our analysis fall under our responsible disclosure policy.

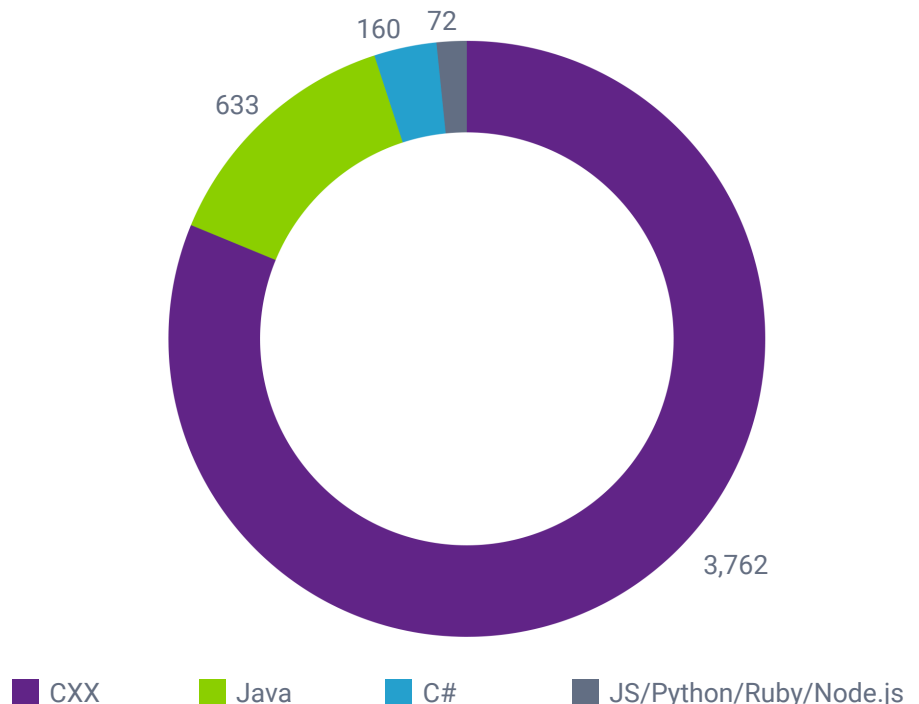
When we began in 2006, we had a total of 120 OSS projects. When this report was written, we had **over 4,600 active OSS projects**, and that number continues to rise.

When this report was written, we had **over 4,600 active OSS projects.**

Project breakdown by language

From the beginning, Scan has supported both the C and C++ languages. In 2012, we introduced Java. In subsequent years, C#, JavaScript, Python, Ruby, and Node.js were added to support the proliferation of these languages in the OSS community. While C/C++ dominates the active projects in Scan, we have a sizable representation of Java projects. Low C# adoption may have to do with its initial role in the Microsoft ecosystem. For JavaScript, Python, Ruby, and Node.js, the low adoption rates are likely due to a lack of awareness regarding the recent availability of these languages in Scan.

Number of projects



Scan today

(continued)

Lines of code per language

Unsurprisingly, C/C++ projects contribute the most in terms of managed lines of code within Scan. Collectively, the total lines of code associated with all languages in active projects is approximately **760 million**.

The total lines of code in active projects in Scan is approximately **760 million**.

Language	Millions of lines of code (MLoC)
CXX	728
Java	30
C#	3.7
JavaScript/Python/Ruby/Node.js	3.1

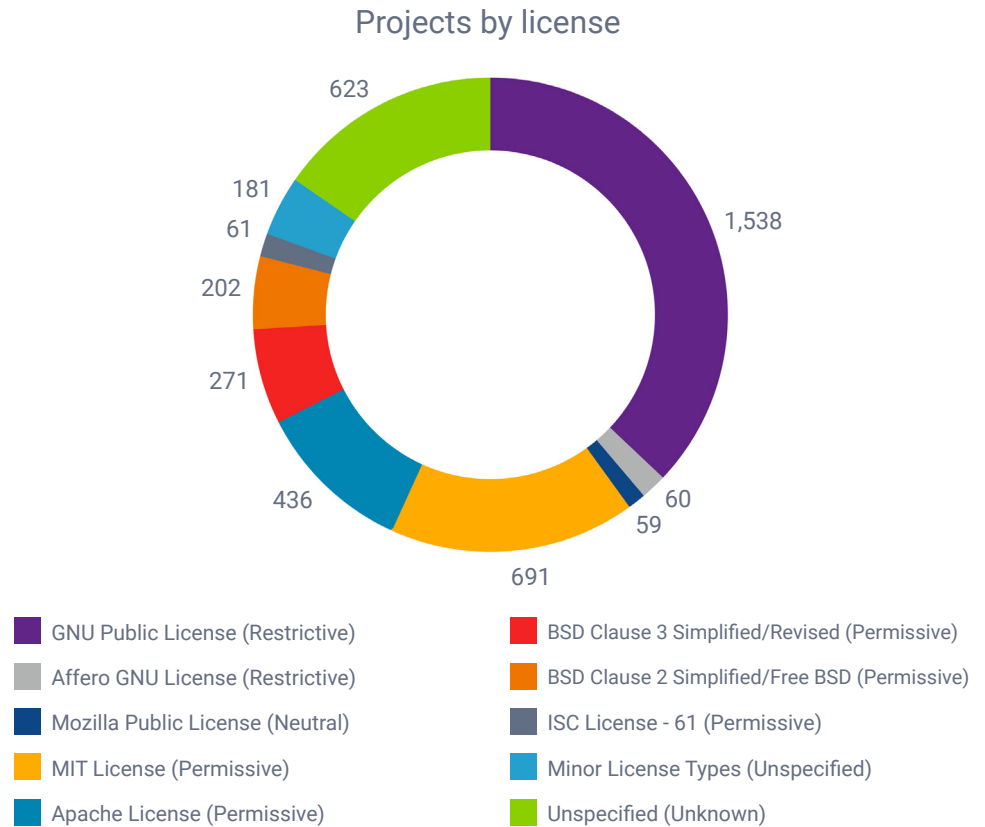
Largest project size by language

Large projects in Scan range from hundreds of thousands to tens of millions of lines of code.

Language	Lines of code
CXX	16,741,683
Java	4,539,805
C#	518,673
JavaScript/Python/Ruby/Node.js	779,720

License distribution

Ignoring minor license types or projects with unspecified licenses, we can divide OSS license adoption between restrictive license types, such as the GNU Public License (GPL), and more permissive licenses, such as the Berkeley Software Distribution (BSD). Among active projects in Scan, we see **restrictive licenses used by 1,597 projects** versus the **1,661 projects that use more permissive licenses**. This suggests that at least within the Scan community, there is roughly equal representation of these types of licenses.



OSS maturity

Looking at active projects within Scan, we can see key behaviors that indicate project maturity. These include frequent analysis (usually associated with Scan integration into the continuous integration / continuous deployment [CI/CD] pipeline of a project), triage of identified defects, active reduction of false positives in analysis results, and configuration of component maps.

Consider the following:

- **The number of active projects** that have **submitted builds for analysis** since the beginning of January 2016 is 4,117. Of those projects, nearly **50% (2,049)** do so using Travis CI. This suggests that active projects use some form of CI/CD and highlights the success of our GitHub / Travis CI integration, which began in 2013.

“**Travis CI** is a hosted, distributed, continuous integration service used to build and test software projects hosted at GitHub. Open source projects may be tested at no charge via travis-ci.org.”

—Wikipedia

- **The number of active projects** that have been **triaged** since the beginning of January 2016 is **2,509**. Viewing and classifying issues requires a high degree of developer engagement, as static analysis is not foolproof and requires developers with intimate knowledge of the codebase to verify identified defects.
- **The number of active projects** that are configured to make use of **modeling** is **1,120**. False positives (FPs) are a characteristic of all static analysis tools. An alternative way of reducing FPs introduced in analysis results is by implementing a modeling file within a project. This helps guide our analysis to do the correct thing when it encounters coding constructs that may be unique to the codebase. Not all projects require modeling, but for those that do, modeling provides a mechanism for improving the quality of their analysis results.

Scan today

(continued)

Percentage of projects configured with modeling

Language	Percentage of projects using modeling
CXX	27.09%
Java	12.32%
C#	0.00%
JavaScript/Python/Ruby/Node.js	8.33%

- **The number of active projects** that make use of **component maps** is **3,216**. The configuration of component maps, by which developers can identify which parts of the code are affected by defects, is another behavior that reflects increasing maturity. In particular, the use of components allows developers to visualize hot spots within their codebases that may have higher than normal defects. This in turn allows them to prioritize their fixes accordingly.

Percentage of projects configured with components

Language	Percentage of projects w/ components
CXX	69.94%
Java	64.14%
C#	75.63%
JavaScript/Python/Ruby/Node.js	51.39%

Together, these numbers show significant adoption of secure software development practices across the active OSS projects within Scan.

A note on false-positive rates

In the context of static analysis, FPs occur when the analysis engine incorrectly identifies a defect as being present when the code is in fact correct. Reducing FPs allows actual defects to be more easily identified by developers.

High FP rates in tools lead to more issues to investigate and remediate. This can hamper adoption as developers lose confidence in the ability of the tool to find actual defects. In previous Scan reports, we provided metrics regarding the FPs encountered in Scan. By looking at the total number of dismissed defects, we can approximate the FP rate from 2013 (when Scan underwent a significant re-architecture to the current system) to the present.

Year	FP rate
2008	13.3%
2009	9.8%
2012	9.7%
2013	10.2%
2017	9.7%

*Note: Only years where FP rates are known are included.

Scan today

(continued)

The relatively high fidelity of the results provided by Scan against a wide variety of different codebases suggests that identified issues are close to 90% actionable.

Scan's community

In 2013, there was a significant re-architecture of Scan. This prevents us from knowing precisely how many users actively engaged with the previous incarnation, but we do have user statistics from that point onward.

The **total number of active users is 21,191**. A large majority of Scan's users have associated GitHub accounts (69%). Roughly 44% of active users have viewed defects associated with their projects.

The total number of active users of Scan is **21,191**.

When we look at user activity from **January 2016** onward, we see that **11,950** users logged in to Scan (approximately 56% of all active users). Of these, roughly 56% had GitHub accounts, and 34% viewed defects.

Access to projects in Scan is governed by roles associated with users. These can be broken down with the following privileges:

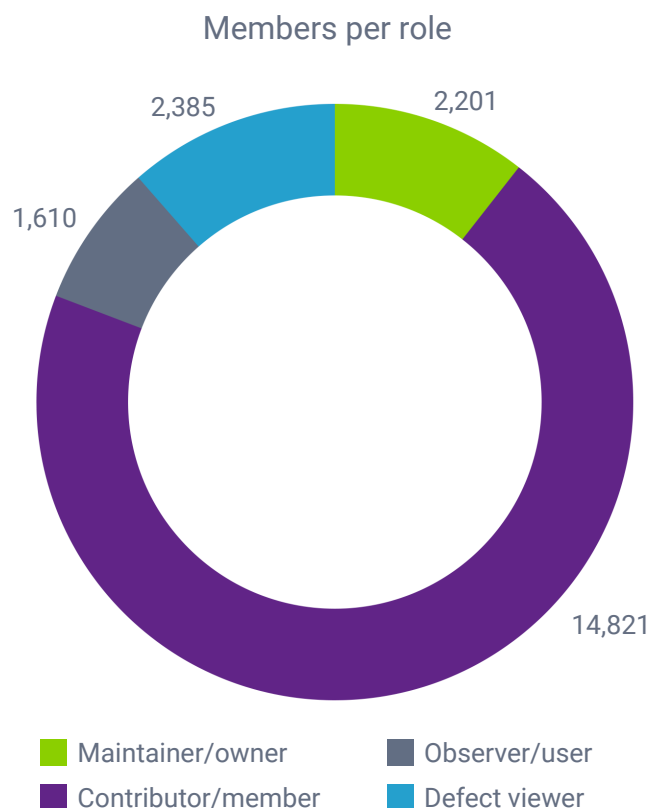
- **Observer/user:** This role grants the ability to view a project's summary/overview.
- **Defect viewer:** In addition to observer privileges, this role grants the ability to view defects associated with a project. Access to defects is read-only.
- **Contributor/member:** This role grants full visibility to the project and the ability to triage defects.
- **Maintainer/owner:** In addition to contributor/member privileges, this role grants the ability to submit builds on behalf of the project as well as manage membership.

Looking at the distribution of project members, we can see that 80% are directly affiliated with a project, being either contributor/members or maintainer/owners.

Scan today

(continued)

Distribution of project members vs. roles



We can conclude from these user metrics that Scan's most active users are a subset of an OSS project's community. This is unsurprising since a high degree of familiarity with the codebase is required to interpret analysis results. These numbers also support Robert Glass' observation that the rate at which bugs are introduced far exceeds the availability of qualified/experienced reviewers. When we consider the roughly 400,000 outstanding defects in Scan versus the 600,000+ fixed defects, the ability to accurately identify high-impact defects becomes even more important. The accuracy of our analysis, as measured by our low false-positive rate, allows for fewer developers to be highly effective in identifying and prioritizing critical fixes across the OSS ecosystem.

Improvements to Scan

Language support

We've slowly been expanding language support to the point where we can provide analysis for the following languages:

- C/C++
- Java
- C#
- JavaScript
- Python
- Ruby
- Node.js

Scan today (continued)

In addition, with the prevalence of multilanguage codebases, we have the ability to capture JavaScript, Python, Ruby, and Node.js as part of the main capture process. For example, if you are using Java and JavaScript, you could specify the capture process like this:

```
# cov-build --fs-capture-search mydir --fs-capture-search-exclude-regex  
".*\\.java" javac Test.java
```

This feature is particularly useful for web applications.

Update policy

We are committed to providing the most current major release of our commercial tools to Scan users. As part of our platform and tools update policy, we support only the last three major releases of our tools. This means projects are expected to update their tools approximately once a year (or more frequently if they want the latest features/support). The current supported versions are as follows:

- 8.5.0.x (to be retired January 2018)
- 8.7.0.x
- 2017.07 (latest)

Analysis submission increases

We've upgraded the back-end infrastructure to accommodate the growth in newly submitted projects. Existing projects should see a significant improvement in the turnaround time of their analyses. Our upgrades have allowed us to increase the build submission limits as follows:

- 4 builds per day, 28 per week, for projects with fewer than 100,000 lines of code
- 3 builds per day, 21 per week, for projects with 100,000–500,000 lines of code
- 2 builds per day, 14 per week, for projects with 500,000–1 million lines of code
- 1 build per day, 7 per week, for projects with more than 1 million lines of code

Build submission

For projects that are hosted on GitHub, the easiest way to include Scan in your CI/CD process is to integrate with Travis CI. One of the key advantages of this process is that it will automatically use the latest version of our tools without any intervention—a great way to ensure that your project benefits from any analysis improvements or bug fixes.

For projects not hosted on GitHub, as an alternative, Eric Raymond (known by his eponymous abbreviation of ESR in the OSS community) has contributed a tool to the community, called `coverity-submit`, that can be used to simplify submission to Scan (<http://www.catb.org/~esr/coverity-submit/>).

Attribution

As part of our ongoing efforts to quantify Scan's impact on the quality and security of OSS, we would appreciate attribution of defects/issues identified with Scan. This can be done by adding a small source code comment that includes the following information:

- Coverity CID
- Date fixed
- Email of project contributor submitting the fix

CASE STUDY:

Quality, security, and project maturity

On Oct. 7, 2017, Google published a security advisory regarding Dnsmasq, a DNS forwarder and DHCP server packaged in distributions such as Ubuntu and used in Android and Kubernetes. In total, the advisory disclosed seven vulnerabilities that could be used in combination to compromise a system via remote code execution. Two vulnerabilities in particular are interesting from the perspective of Scan:

“CVE-2017-14493 is a trivial-to-exploit DHCP-based, stack-based buffer overflow vulnerability. In combination with CVE-2017-14494 acting as an info leak, an attacker could bypass ASLR and gain remote code execution.”

—<https://security.googleblog.com/2017/10/behind-masq-yet-more-dns-and-dhcp.html>

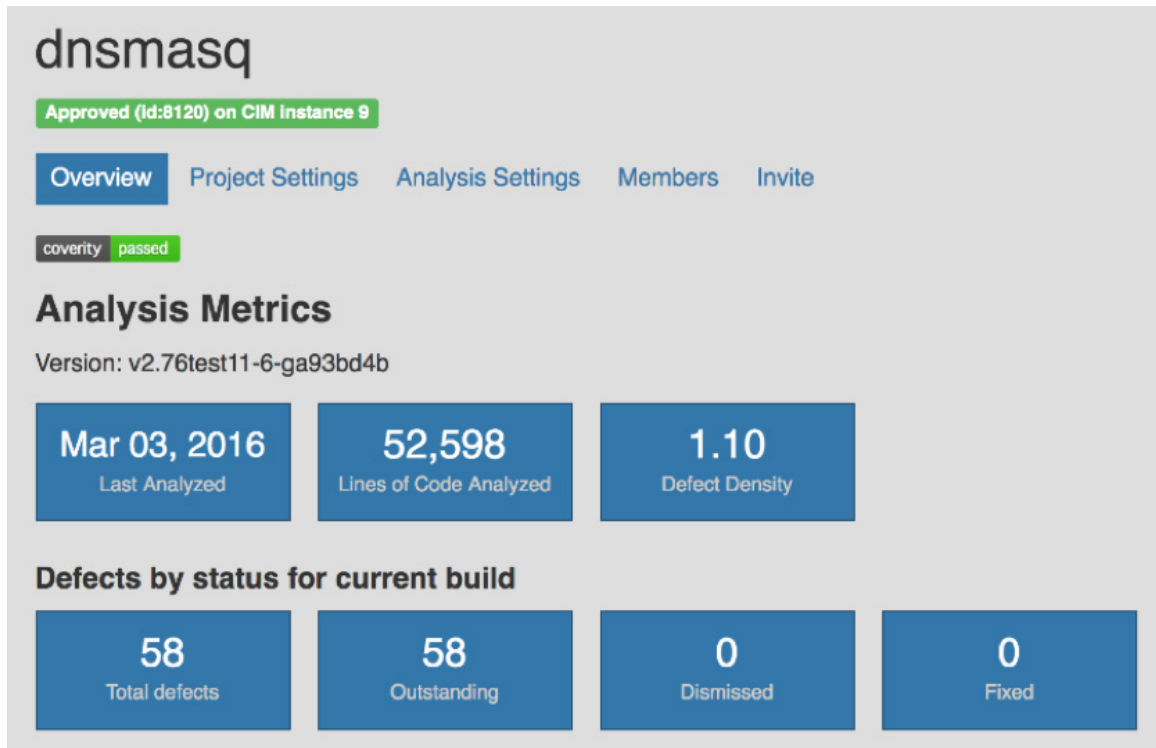
As part of the disclosure, Google provided an address sanitizer report for CVE-2017-14493, which gives us an idea of where this vulnerability can be found in the code:

```
==33==ERROR: AddressSanitizer: stack-buffer-overflow on address
0x7ffcbeef81470 at pc 0x0000004b5408 bp 0x7ffcbeef81290 sp 0x7ffcbeef80a40
WRITE of size 30 at 0x7ffcbeef81470 thread T0
#0 0x4b5407 in __asan_memcpy (/test/dnsmasq/src/dnsmasq+0x4b5407)
#1 0x575d38 in dhcp6_maybe_relay /test/dnsmasq/src/rfc3315.c:211:7
#2 0x575378 in dhcp6_reply /test/dnsmasq/src/rfc3315.c:103:7
#3 0x571080 in dhcp6_packet /test/dnsmasq/src/dhcp6.c:233:14
#4 0x544a82 in main /test/dnsmasq/src/dnsmasq.c:1061:2
#5 0x7f93e5da62b0 in __libc_start_main (/lib/x86_64-linux-gnu/libc.
so.6+0x202b0)
#6 0x41cbe9 in _start (/test/dnsmasq/src/dnsmasq+0x41cbe9)
```


CASE STUDY (CONTINUED):

Quality, security, and project maturity

As it stands, Dnsmasq has been analyzed by Scan. This is what we know about the project:



This information reveals that while Dnsmasq consists of roughly 50,000 lines of code, the defect density remains high (greater than 1), with a handful of outstanding defects. In addition, the last time this project was analyzed was March 2016. This information suggests a project that poses a risk for consumers.

CASE STUDY (CONTINUED): Quality, security, and project maturity

In addition, when looking at the outstanding defects, we find the following:

The screenshot displays the Coverity Scan interface. At the top, there is a navigation bar with 'Configuration', 'Return to Dashboard', 'Guided Tour', 'Help', 'coverity_admin', and 'Enter GDB'. Below this is a table of issues. The table has columns for CID, Type, Impact, Status, First Detected, Owner, Classification, Severity, Action, and Component. Three issues are listed:

CID	Type	Impact	Status	First Detected	Owner	Classification	Severity	Action	Component
72206	Out-of-bounds access	High	New	03/03/16	Unassigned	Unclassified	Unspecified	Ignore	Other
72187	Buffer not null terminated	High	New	03/03/16	Unassigned	Unclassified	Unspecified	Ignore	Other
72186	Buffer not null terminated	High	New	03/03/16	Unassigned	Unclassified	Unspecified	Undecided	Other

Below the table, there is a section for '1 of 58 issues selected'. The main area shows a code editor for 'rfc3315.c'. The code includes several lines of C code, with annotations and warnings. A red warning box highlights a buffer overflow issue:

```

218 state->mac_len = opt6_len(opt) - 2;
219
220 CID 72206 (#1 of 1): Out-of-bounds access (OVERRUN)
221 26. overrun-buffer-arg: Overrunning buffer pointed to by &state->mac[0] of 16 bytes by passing it to a function which accesses it at byte offset 4294967293
    state->mac_len (which evaluates to 4294967294).
222 memcopy(&state->mac[0], opt6_ptr(opt, 2), state->mac_len);
223 }
224
225 for (opt = opts; opt = opt6_next(opt, end))
226 {
    int o = new_opt6(opt6_type(opt));
  
```

On the right side of the interface, there is a sidebar with a title '72206 Out-of-bounds access' and a description: 'Access of memory not owned by this buffer may cause crashes or incorrect computations. In dhcp6_maybe_relay: Out-of-bounds access to a buffer (CVE-119)'. Below this are several expandable sections: 'Triage', 'Projects & Streams', 'Detection History', 'Triage History', and 'Occurrences'.

The buffer overflow at the root of **CVE-2017-14493**, which is used in conjunction with an information leakage flaw in **CVE-2017-14494**, was identified over a year before—meaning the vulnerability could have been prevented if the guidance in Scan had been acted on.

This vulnerability gives us some insight into how current program weaknesses can become future vectors of exploitation. Improvements in software quality via static analysis can have a direct impact on the overall security posture of OSS—but only if the projects themselves adopt more mature software development practices.

From defect density to maturity

In the past, we focused primarily on defect density as a means of measuring code quality. We even developed the concept of integrity levels, which specified criteria for increasing levels of code quality.

- **Level 1** requires the software to have less than or equal to 1 defect per thousand lines of code, which is approximately the average defect density for the software industry.
- **Level 2** requires the software to have 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 1-million-line codebase would have to have 100 or fewer defects to qualify for Level 2.
- **Level 3** is the highest bar in the rating system today. All three of the following criteria must be met:
 - i. A defect density less than or equal to 0.01 per thousand lines of code (defect density ≤ 0.01 defect/KLoC), which is approximately in the 99th percentile for the software industry. This means that a 1-million-line codebase must have 10 or fewer static analysis defects remaining. The requirement does not specify 0 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 make achieving a target Level 3 for the release nearly impossible).
 - ii. False positives constitute less than 20% of the results or else audited by Coverity. A higher false-positive rate indicates either misconfiguration, usage of unusual idioms, or incorrect diagnosis of a large number of defects. Coverity Static Analysis has less than 20% false positives for most codebases, so we reserve the right to audit false positives when they exceed this threshold.
 - iii. Zero defects marked as major severity by the user. In the Coverity user interface, users can set the severity of each defect by setting an attribute 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, no major defects should remain unfixed to achieve Level 3.

While these criteria could be applied to proprietary closed source codebases backed by the development resources to commit to this effort, they do not represent the reality of OSS development. For example, Linux, which has been analyzed in Scan since 2006, has a defect density of 0.47, which means that in the last 10+ years, it has managed to reach only Level 1—despite being one of the most broadly adopted pieces of OSS.

Only a few OSS projects ever attain Level 2 or higher. PostgreSQL is one example:

- PostgreSQL 8.4: 0.00 (end-of-life)
- PostgreSQL 9.0: 0.06 (end-of-life)
- PostgreSQL 9.1: 0.05 (end-of-life)
- PostgreSQL 9.2: 0.06
- PostgreSQL 9.3: 0.06
- PostgreSQL 9.4: 0.04
- PostgreSQL 9.5: 0.05
- PostgreSQL 9.6: 0.10

Today, OpenSSL has a defect density of 0.3 despite a concerted effort to improve the codebase post-Heartbleed. Supported by the Linux Foundation's Core Infrastructure Initiative (CII), the project provided funding for developers as well as a full audit of the codebase. While OpenSSL is arguably more secure today due to the improvement of the quality of its code, its defect density does not fully reflect the recent improvement in the project's maturity.

Defect density can provide an approximation regarding the quality of OSS, but by itself, it is insufficient to evaluate the maturity of a project and the risk associated with consumption and usage of software. For that, we need to look a little deeper.

The future (continued)

The Census Project

The CII is the combined efforts of the Linux Foundation and industry leaders such as Amazon Web Services, Facebook, Google, IBM, and Microsoft to collaboratively identify and fund critical open source projects in need of assistance. In 2015, the Census Project was initiated with the Institute for Defense Analyses (IDA) and funded jointly by the CII and the U.S. Department of Homeland Security Homeland Open Security Technology (DHS HOST) program at the Georgia Tech Research Institute (GTRI).

The Census Project's report was limited in scope (it focused only on the analysis of software packages in Debian's core distribution) and provided results for a single point in time. Nonetheless, its approach to assessing risk expanded awareness of factors other than defect density that can be used to evaluate the maturity of an OSS project. Scan was mentioned as a possible source of potentially useful security metrics in the "Open Source Software Projects Needing Security Investments" paper by the IDA (https://www.coreinfrastructure.org/sites/cii/files/pages/files/pub_ida_if_cii_070915.pdf).

The identification algorithm used by the Census Project considered the following criteria when evaluating OSS projects and applied an associated risk index (taken from the Census Project site at <https://www.coreinfrastructure.org/programs/census-project>):

- **Website:** If the project has no website, it receives 1 point.
- **Common Vulnerabilities and Exposures (CVEs):** If the project has four or more CVEs (since 2010), it receives 3 points; two or three CVEs, 2 points; and one CVE, 1 point. Note that the absence of CVEs does not necessarily indicate the absence of vulnerabilities; it may instead indicate that no one has looked for vulnerabilities in the project or that no one filed the CVE requests for vulnerabilities when they were found.
- **Contributor Count:** If the 12-month contributor count is zero, the project receives 5 points; one to three contributors, 4 points; and unknown contributors, 2 points.
- **Popularity:** If the package is in the top 1% of installed packages tracked by Debian, it receives 2 points, or 1 point if it is in the top 5%.
- **Main Language:** If the project's main language is C or C++, add 2 points.
- **Network Exposure:** If the package is directly exposed to the network (whether client or server), it receives 2 points. If it is used to process data provided by a network, it receives 1 point. It receives 1 point if it typically runs as root (either via suid or directly) or controls access to such and therefore is a risk for local privilege escalation.
- **Application Data Only:** The package gets 3 points taken away if the Debian database reports that it is application data or stand-alone data rather than an application.

In addition to the criteria above, a few others were discussed:

- **Dependencies:** Add 1 point to the package if one to five other packages depend on it, or 2 points if more than five. This parameter would promote packages that are often relied on by other packages. In doing so, it would identify core infrastructure. This parameter may have some overlap with the Popularity parameter already included.
- **Patches:** If the deb or rpm includes more than five patches that have not been accepted upstream, the package receives 1 point. Distros carry patches for unique packaging requirements, and when the upstream project is nonresponsive, the patches are often less reviewed than the original project and so may add risk to the project. This parameter may have some overlap with the Contributor Count parameter already included.
- **ABRT Crash Statistics:** If the crash statistics are increasing over time, add 2 points to the project's score. If the statistics are stable but high, add 1 point.

The future (continued)

The Census Project's focus on the technical dimensions of an OSS project is another perspective with which to measure maturity.

Community Health Analytics Open Source Software

Risk extends beyond technical measures; it encompasses factors such as community engagement and health. In addition to the Census Project, the Linux Foundation has also launched Community Health Analytics Open Source Software (CHAOSS), which focuses on analytics related to OSS communities.

Generally, when a project's health is being measured, commits to the source code repository are used as a proxy for health. This is problematic because commit volume may indicate different things. For example, low commit volume may suggest either lack of developer engagement or a mature project with few issues. On the other hand, high commit volume may suggest high developer engagement but may also be a sign of low-quality commits, which may destabilize a project and introduce defects or vulnerabilities. Commits, like static analysis defects, can be used as a health indicator, but without sufficient context, they can be misleading and incomplete.


Other metrics that may help assess the maturity of a project through its community are as follows:

- Bus factor (or too few developers)
- Issue response rate
- Increasing backlog of bugs
- Decreasing maintainer activity
- Velocity of maintainer onboarding
- Decrease in contributions from the community
- Longevity of active maintainers

None of these by itself is an indicator of a project in jeopardy, but a number of them together increases the probability of risk and provides additional insight into the maturity of the project. To find out more, see <https://chaoss.community/>.

A new view of maturity

Maturity needs context to be meaningful, and when metrics are expanded beyond defect density to include additional technical dimensions, as suggested by the Census Project, and metrics regarding community health, consumers of OSS can have a fuller understanding of the risks in the software they rely on.



Scan is proof that it takes only a few developers to make a significant improvement.

Conclusion

Since its inception, Scan has enabled developers to fix over 600,000 defects across some of the most important projects in open source. As part of that effort, it has also helped improve the maturity of the software development practice of active OSS contributors by supporting the continuous integration of analysis results and the accurate identification of discovered issues. The effectiveness of Scan's static analysis is reflected in the low false-positive rate of under 10% over 700 million lines of code currently managed by Scan. Given the modest number of developers versus the relative size of the individual source codebases, Scan is proof that only a few developers are required to make a significant improvement to the entire OSS ecosystem. The accuracy of our results translates directly into actionable developer guidance.

The accuracy of our results translates directly into actionable developer guidance.

Approaching project maturity from the perspective of static analysis leads us to measure improvements to OSS projects using the metric of defect density. While this provides some useful information regarding improvements in the quality of code, it is far from complete. From a broader perspective of maturity, we need to consider additional metrics. Technical criteria suggested by the CII's Census Project is one starting point. But in addition to the technical aspects, community health should be considered. This is the goal of the CHAOSS project.

It is becoming crucial to be able to assess risks associated with the consumption of OSS. The potential to provide a holistic view of software risk and maturity by combining information from multiple dimensions will be essential as OSS becomes ever more pervasive in technology.

THE SYNOPSYS DIFFERENCE

Synopsys offers the most comprehensive solution for building integrity—security and quality—into your SDLC and supply chain. We've united leading testing technologies, automated analysis, and experts to create a robust portfolio of products and services. This portfolio enables companies to develop customized programs for detecting and remediating defects and vulnerabilities early in the development process, minimizing risk and maximizing productivity. Synopsys, a recognized leader in application security testing, is uniquely positioned to adapt and apply best practices to new technologies and trends such as IoT, DevOps, CI/CD, and the Cloud. We don't stop when the test is over. We offer onboarding and deployment assistance, targeted remediation guidance, and a variety of training solutions that empower you to optimize your investment. Whether you're just starting your journey or well on your way, our platform will help ensure the integrity of the applications that power your business.

For more information go to www.synopsys.com/software.

SYNOPSYS®

185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

U.S. Sales: **800.873.8193**

International Sales: **+1 415.321.5237**

Email: **sig-info@synopsys.com**