coverity®

# Coverity Scan

## Project Spotlight: Python

# Coverity Scan Service

The Coverity Scan™ service 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. With more than 450 projects participating, the Coverity Scan service enables open source developers to scan–or test–their Java, C and C++ code as it is written, flag critical quality and security defects that are difficult (if not impossible) to identify with other methods and manual reviews, and provide developers with actionable information to help them to quickly and efficiently fix the identified defects. (More than 20,000 defects identified by the Coverity Scan service were fixed by open source developers in 2012 alone.) The Coverity Scan service is powered by our award-winning Coverity SAVE® static analysis verification engine, which applies multiple patented techniques for highly accurate defect detection.

We've expanded beyond our annual Coverity Scan Report to create a series of open source project spotlights. This spotlight features Python, an innovative, open source programming language which has recently achieved the highest level of quality in the Coverity Scan service. In order to reach this milestone, all three of the following criteria were met:

- Defect density that is 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 remaining defects.
- False positive rate that is less than 20%.
- Zero defects marked as "Major Severity" by the user.

# Introduction to Python

Python is a powerful, dynamic programming language that is used in a wide variety of application domains. Some of its key distinguishing features include very clear and readable syntax; strong introspection capabilities; intuitive object orientation; natural expression of procedural code; full modularity, supporting hierarchical packages, exception-based error handling, very high level dynamic data types, extensive standard libraries and third party modules for virtually every task; extensions and modules easily written in C, C++ (or Java for Jython, or .NET languages for IronPython); and the ability to be embedded within applications as a scripting interface. Python has a robust standard library and has a highly optimized byte compiler and support libraries. The project is available for all major operating systems.

Python has a very open and collaborative spirit. In fact, the Python Core mentorship project (http://pythonmentors.com/) contains the following quote from Winston Churchill at the top of its site, "We make a living by what we get, we make a life by what we give." The mission of the project is to connect anyone interested in contributing to Python Core development with members of the development team; it is based on the idea that the best way to welcome new people into any project is to provide a venue which connects them to a variety of mentors who can assist in guiding them through the contribution process, including discussions on lists such as python-dev, python-ideas, the bug tracker, mercurial questions, code reviews, etc. Python is also committed to diversity (http://www.python.org/community/diversity), as the team believes it makes the community stronger and creates more vibrancy, which in turn creates opportunities for more contributors – and more sources for ideas.

The Pyladies program (http://www.pyladies.com) is one example of Python's commitment to diversity. It is an international mentorship group focused on helping more women become active participants and leaders in the Python community. The Pyladies program's mission is to promote, educate and advance a diverse Python community through outreach, education, conferences, events and social gatherings.

It is in part due to this dedication to diversity, mentorship and grooming of new participants, that the Python project is able to achieve the highest level of code quality and enable even the newest members to add value to the project.

## Python: Then and Now

Python has been an active project in the Coverity Scan service since 2006.

**PYTHON ANALYSIS: 2006 TO 2013**

| Year | Version(s) | Lines of Code Analyzed | New Defects Identified | Defects Fixed |
|---|---|---|---|---|
| 2006 | 2.5 | 279,518 | 169 | 153 |
| 2007 | 2.5 | 286,591 | 58 | 16 |
| 2008 | 3.0 | 311,626 | 32 | 44 |
| 2009 | 3.1 | 328,294 | 15 | 11 |
| 2010 | 3.1 | 329,651 | 1 | 1 |
| 2011 | 3.2 | 340,911 | 327 | 93 |
| 2012 | 3.3 | 392,773 | 213 | 264 |
| 2013 to date | 3.3.2 | 396,041 | 181 | 278 |

Since 2006, the Coverity Scan service has identified 996 new defects and the Python community has fixed 860 of these defects. The year 2010 was a bit of an anomaly for the Python project in the Scan service. Due to some transitions in the program, regular analysis was not performed on the code. However, since 2011, we have been able to address the issues and resume more consistent analysis of the Python code. The higher number of defects in 2011 reflects advances in the Coverity® static analysis technology that enable detection of a broader range of defects versus a statement of declining quality. Compared to 99% of all software projects, Python has extremely low defect density which reflects their commitment to quality.

| DEFECT DENSITY BY PROJECT SIZE COMPARISON | | | |
|---|---|---|---|
| **Lines of Code** | **Python** | **Open Source** | **Proprietary** |
| 100,000-499,999 | Less than .01 | .60 | .66 |

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

For example, Py_FatalError() never returns, and the analysis was finding infeasible defects since it could not understand this from the code. Python added a model for Py_FatalError() which marks this function as a "killpath" – effectively terminating code paths that reach this function and eliminating those false positives. They provide additional models for other interesting behaviors, such as which functions can never return a NULL value, which functions can tolerate tainted input parameters, and so forth–leading to fewer false positives and more true positives.

The following are the specific types of defects outstanding in Python:

| DEFECTS OUTSTANDING BY TYPE AND IMPACT AS OF AUGUST 15, 2013 | | | |
|---|---|---|---|
| Error handling issues | 1 | Medium | .66 |

Python has no high-impact defects, as required to achieve a Level 3 rating. They have worked very hard to eliminate the key defects in their code over the past eight months. As of December 2012, they had the following high and medium defects, 99% of which have been addressed:

| DEFECTS OUTSTANDING BY TYPE AND IMPACT AS OF DECEMBER 31, 2012 | | |
|---|---|---|
| Memory – corruptions | 20 | High |
| Memory – illegal accesses | 15 | High |
| Resource leaks | 10 | High |
| API usage errors | 1 | Medium |
| Concurrent data access violations | 1 | Medium |

| | | |
|---|---|---|
| Control flow issues | 20 | Medium |
| Error handling issues | 33 | Medium |
| Incorrect expression | 9 | Medium |
| Insecure data handling | 5 | Medium |
| Integer handling issues | 8 | Medium |
| Null pointer dereferences | 14 | Medium |
| Code maintainability issues | 1 | Low |
| Security best practices violations | 10 | Low |
| **Grand Total** | **147** | |

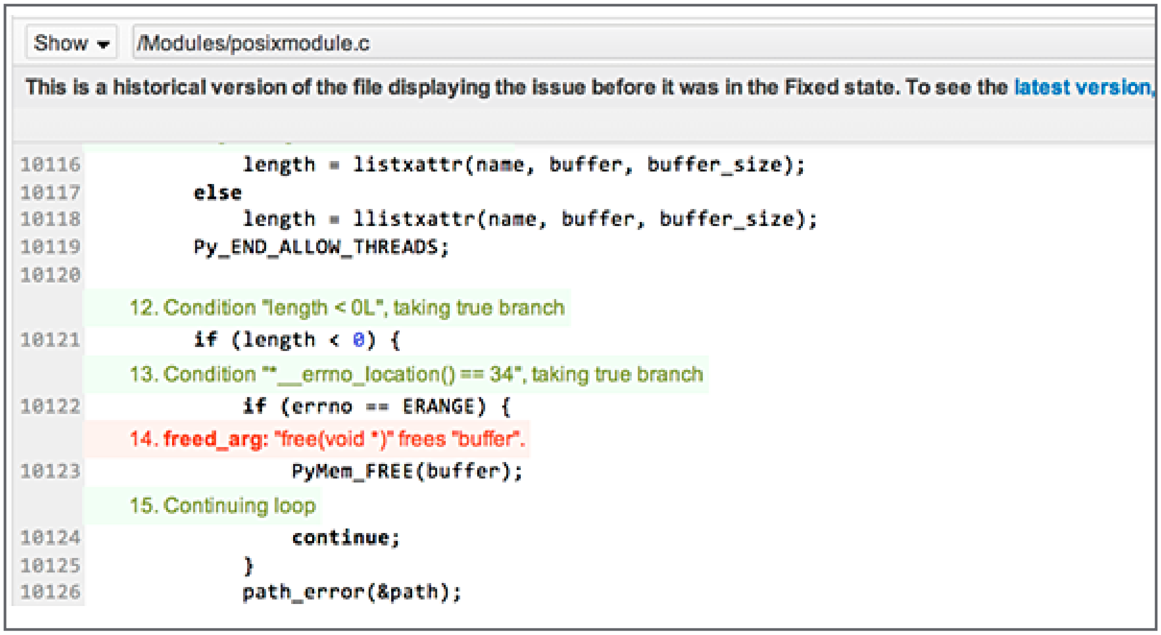
One of the issues the Python team recently fixed through the Coverity Scan service was a double free defect. This particular defect occurs when the program misuses the memory management functions, causing Python to reuse memory that it isn't supposed to use anymore. This can lead to memory corruption, which a smart attacker may be able to leverage to gain control of the computer. In the following image, we see:

Memory was allocated for variable "buffer".

At line 10123, memory was deallocated for variable "buffer".

At line 10161, memory was again deallocated for variable "buffer", but after NULL check.

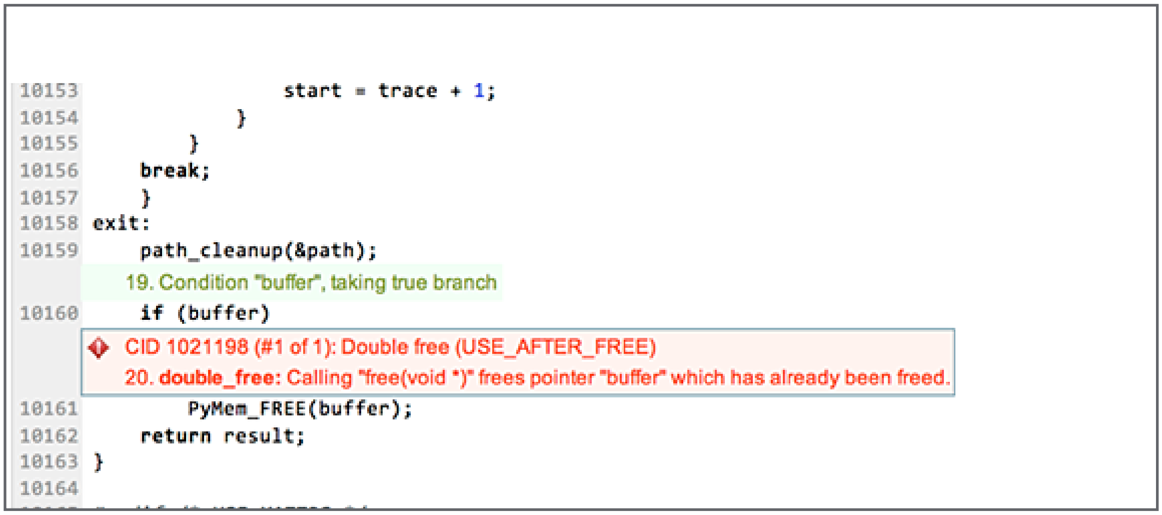
However, variable was never set to NULL after memory being deallocated, which caused memory being deallocated twice for the same variable (called a “double free” defect), and would have resulted in memory corruption.



/Modules/posixmodule.c

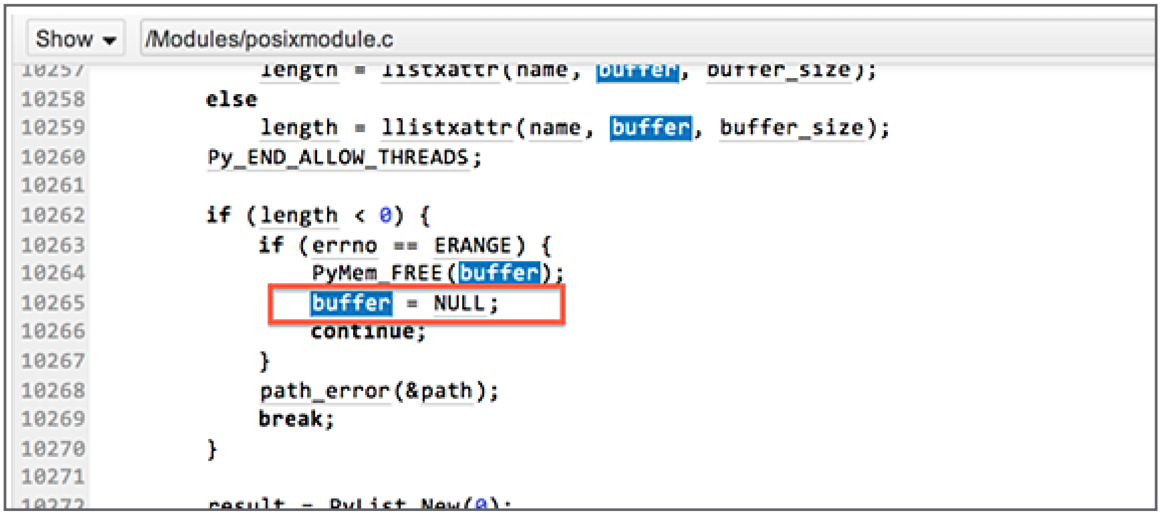
**This is a historical version of the file displaying the issue before it was in the Fixed state. To see the latest version,**

```
            length = listxattr(name, buffer, buffer_size);
        else
            length = llistxattr(name, buffer, buffer_size);
        Py_END_ALLOW_THREADS;

    12. Condition "length < 0L", taking true branch
        if (length < 0) {
    13. Condition "*__errno_location() == 34", taking true branch
            if (errno == ERANGE) {
    14. freed_arg: "free(void *)" frees "buffer".
                PyMem_FREE(buffer);
    15. Continuing loop
                continue;
            }
            path_error(&path);
```



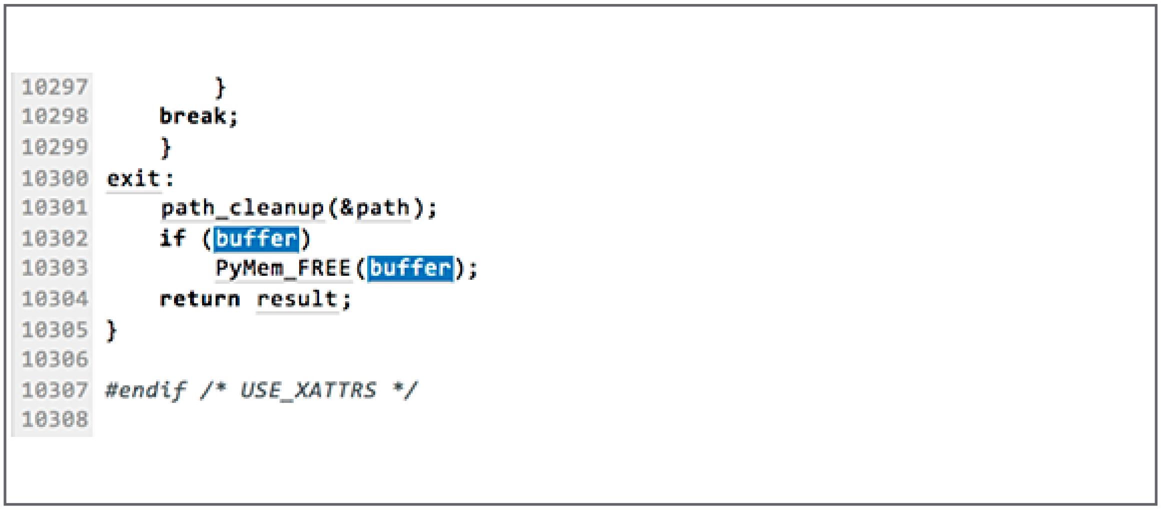
```
                start = trace + 1;
            }
        }
    break;
    }
exit:
    path_cleanup(&path);
    19. Condition "buffer", taking true branch
    if (buffer)
 CID 1021198 (#1 of 1): Double free (USE_AFTER_FREE)
 20. double_free: Calling "free(void *)" frees pointer "buffer" which has already been freed.
        PyMem_FREE(buffer);
    return result;
}

```

After deallocate of memory at line 10123, variable was set to NULL. This change eliminated the double free defect at line 10161.ter:



/Modules/posixmodule.c

```
10257             length = listxattr(name, buffer, buffer_size);
10258         else
10259             length = llistxattr(name, buffer, buffer_size);
10260         Py_END_ALLOW_THREADS;
10261 
10262         if (length < 0) {
10263             if (errno == ERANGE) {
10264                 PyMem_FREE(buffer);
10265                 buffer = NULL;
10266                 continue;
10267             }
10268             path_error(&path);
10269             break;
10270         }
10271 
10272         result = PyList_New(0);
```



```
10297         }
10298     break;
10299     }
10300 exit:
10301     path_cleanup(&path);
10302     if (buffer)
10303         PyMem_FREE(buffer);
10304     return result;
10305 }
10306 
10307 #endif /* USE_XATTRS */
10308 
```

# Q&A with Christian Heimes: A Core Committer to Python since 2007

Q: *Why do you think you are able to achieve high levels of quality?*

A: Python has an established and well working workflow. The majority of commits are accompanied by a ticket. Most bug fixes, except for trivial ones, and all new features are reviewed by other developers before the patches are committed. Documentation updates, change log entries and unit tests are usually part of a patch, too. Large features and modifications go through the PEP (Python Enhancement Proposal) process.

Python core development relies heavily on automatic tests. We have been using buildbot for continuous integration since at least 2006. About 40 buildbot instances are used to run 10k test cases on different of platforms and architectures: Linux (multiple distributions), Windows, Mac, BSD, and even exotic operating systems like Solaris and AIX, and hardware like PPC, MIPS, Sparc and Alpha (Snakebite).

Python uses time-based releases, not feature releases. New features only land in the development branch when they are stable and have been through our review process. We are not under pressure to add "cool stuff" to increase our market share. Our goal is to provide a stable and slowly evolving foundation for our community. Revolutionary pieces of software are developed outside the core by other developers. Some of them are later integrated into the core when they are deemed mature and best practice. Backward compatibility is also very important to us, except when we break it deliberately, such as with Python 3. Most of Python is written in Python, too. It's much easier and less error prone to maintain Python code than C code. The rest of Python is written in well-structured ANSI C (C89) with a well-designed C API and a strong focus on POSIX.

Q: *What is it about the developers on your project that you think enables them to create high-quality code?*

A: All core committers are highly motivated and care deeply for Python. Although we are split up across lots of countries, cultures and time zones, we are able to work together as a team very well. A core developer is elected when he or she can show that they are able to contribute good patches over some time. We are a meritocracy.

Q: *What happens if you have a developer working on the project who submits code that doesn't meet quality expectations?*

A: It rarely happens, as most changes go through a thorough review process before they are committed. Once in a while some issues slip through – after all, we are just humans. Since commits are tightly monitored, such issues are pointed out in a matter of hours, even minutes. Either the issue is sorted out as soon as possible or the commit is reverted.

Q: *What sort of process do you follow to ensure high-quality code?*

A: Python has coding standards for C and Python code. Major changes go through the PEP process, other changes go through a review process. We have stable APIs, ABIs and automated tests, and we utilize continuous integration. However we also have tedious bike shedding[1] discussions on the mailing list.

[1] Bike shedding: Technical disputes over minor, marginal issues conducted while more serious ones are being overlooked.

Q: *Do you have people in formal roles to ensure the quality of the code?*
A: In theory, Python has a hierarchy:

> Guido (Benevolent Dictator for Life) > release manager > expert for module or area of interest
> > core committer > contributor

In practice, this hierarchy is never imposed upon somebody but rather used as a tool to aid the development process. Core committers are responsible for their checkins and do their best to meet our demands in quality. They also help contributors to improve their patches and teach them Python's coding conventions and best practices. Experts for a module or topic are often included in the discussion to get their opinion and to benefit from their knowledge.

Q: *Can you describe how development testing and the Coverity Scan service fit into your release process?*
A: Coverity comes into play when the code base has stabilized and a new, minor release is approaching its release candidate phase. It is especially useful to find issues in unlikely code paths, like error cases that are not reached under ordinary circumstances. A stable code base makes it easier to find and fix the problematic code segments.

Recently I went through untriaged Coverity platform-found issues and either fixed, closed or triaged them all. In the future, I'm planning to fix or report issues as they are detected.

Q: *What tools do you use besides Coverity, and how do they impact your ability to deliver high-quality code?*
A: We use the following tools:

- Mercurial (hg) DVCS
- Roundup issue tracker
- Rietveld code review
- buildbot for CI
- Irker to push buildbot and roundup messages to #python-dev IRC channel
- fusil for fuzzing tests and pyfailmalloc to add random malloc() failures (both created by Python core developer Victor Stinner)
- GCC's gcov
- clang analyzer
- instrumented Python builds (--with-pydebug) with extra checks, asserts and reference leak checks

Resources, sponsoring, software and hardware donations are provided by the IT industry (Microsoft, IBM, HP, Oracle, Google and others.)

Most tools are written in Python, too.  An "eat your own dog food" philosophy and practice helps with our quality efforts.

Q: *What challenges do you face with regard to maintaining high-quality code that are unique to open source and how do you overcome those challenges?*

A: We have lots of issues with patches and not enough time to review/apply them. Since no one is getting paid and we work in our free time, it might take a while before they get applied–however never is often better than a "right now" approach. We also target lots of platforms, even exotic ones like AIX, and with Windows, there is a dichotomy between POSIX-like API and Windows API.

# Conclusion and Next Steps for Coverity Scan

Python has shown a long-term commitment to delivering high-quality software to their constituents. They have achieved the highest level of quality and are almost completely free of high- and medium-impact defects. The quality of their code far outpaces that of like-sized commercial offerings. Their spirit of openness, collaboration and mentorship is also unique, and serves as a great example for other open source projects. We would like to thank the Python team for their continued support and participation in the Coverity Scan Service.

Register your open source project with Coverity Scan or sign up for a free trial to get visibility into the quality and security of your software code.

coverity®

# Software Integrity Report

**Project Name:** Python

**Version:** 3.4dev

**Project Description:** http://python.org/

**Project Details:**

**Lines of Code Inspected:** 396,738

**Project Defect Density:** <0.01

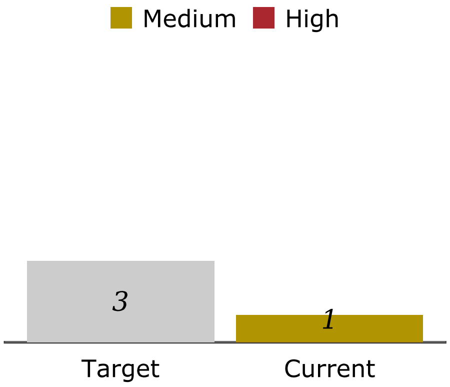
**High-Impact and Medium-Impact Defects:** 1

**Target Level 3** **ACHIEVED**

| | | | |
|---|---|---|---|
| Company Name: | Python | Coverity Product: | Static Analysis |
| Point of Contact: | Coverity Scan | Product Version: | 6.6.1 |
| Client email: | scan-admin@coverity.com | Coverity Point of Contact: | Integrity Report |
| Report Date: | Aug 15, 2013 11:26:06 PM | Coverity email: | integrityrating@coverity.com |
| Report ID: | 96f1dfdc-9c4d-4645-ae45-b186c3a3e229 | | |

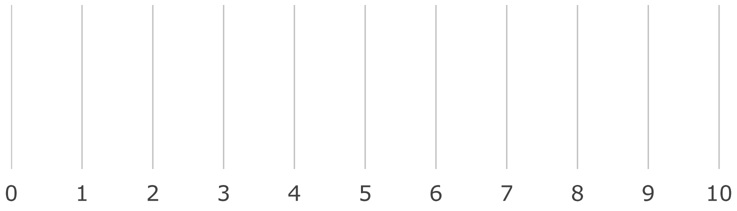
The Coverity Integrity Rating Program provides a standard way to objectively measure the integrity of your own software as well as software you integrate from suppliers and the open source community. Coverity Integrity Ratings are established based on the number of defects found by Coverity® Static Analysis 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.



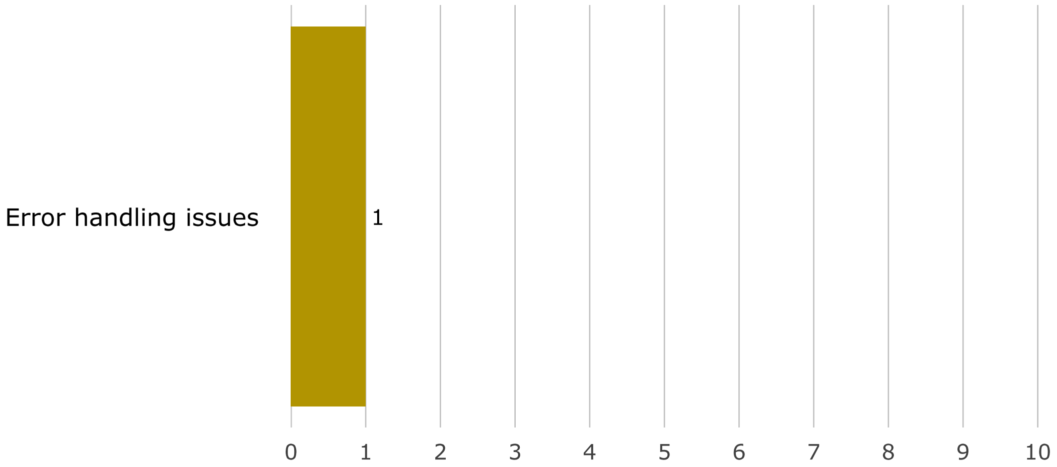
## High-Risk Defects

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



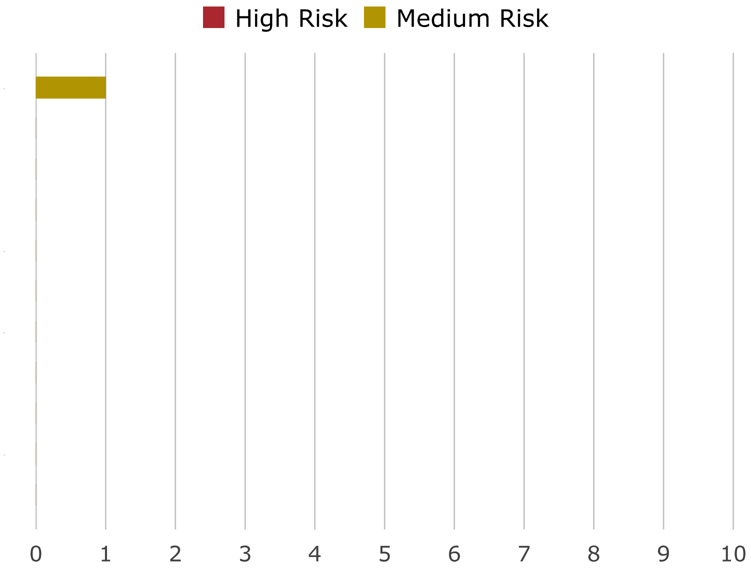
## Medium-Risk Defects

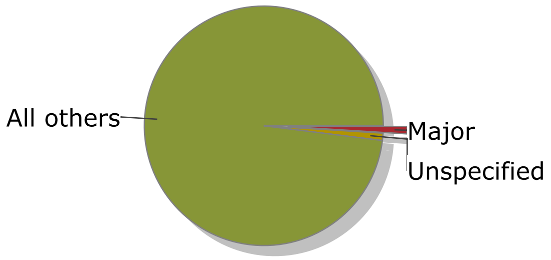
*Medium-impact defects that cause incorrect results, concurrency problems, and system freezes.*



## Defect Risk by Component

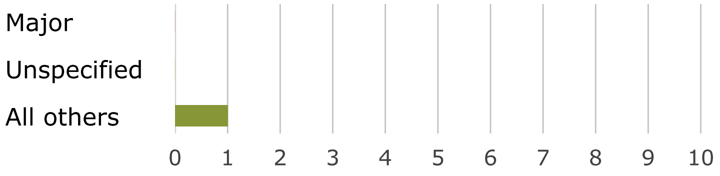
| Component | Owner | Defect Density |
|---|---|---|
| Modules | | <0.01 |
| libffi | | 0.00 |
| ctypes | | 0.00 |
| Include | | 0.00 |
| Other | | 0.00 |
| Parser | | 0.00 |
| usr include | | 0.00 |
| libmpdec | | 0.00 |
| expat | | 0.00 |
| Objects | | 0.00 |
| Python | | 0.00 |





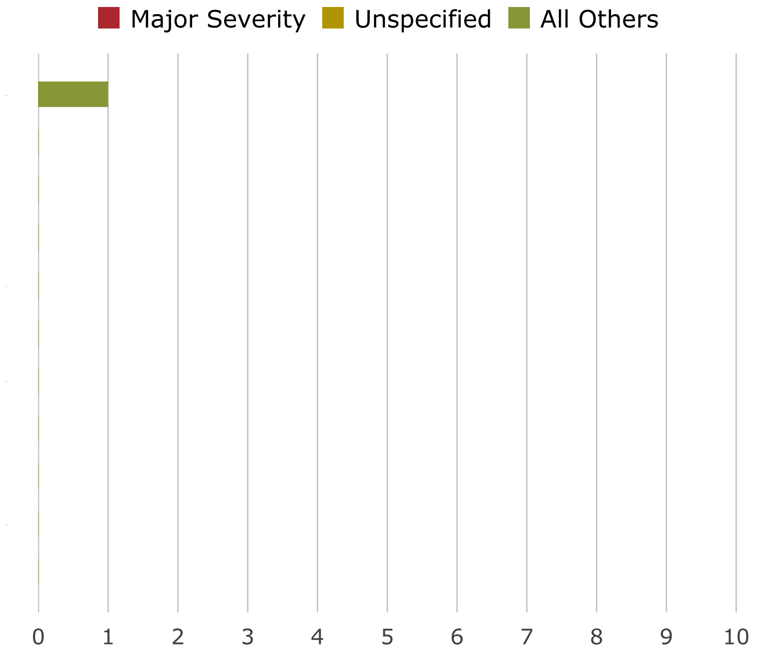
## Defects by Assigned Severity

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

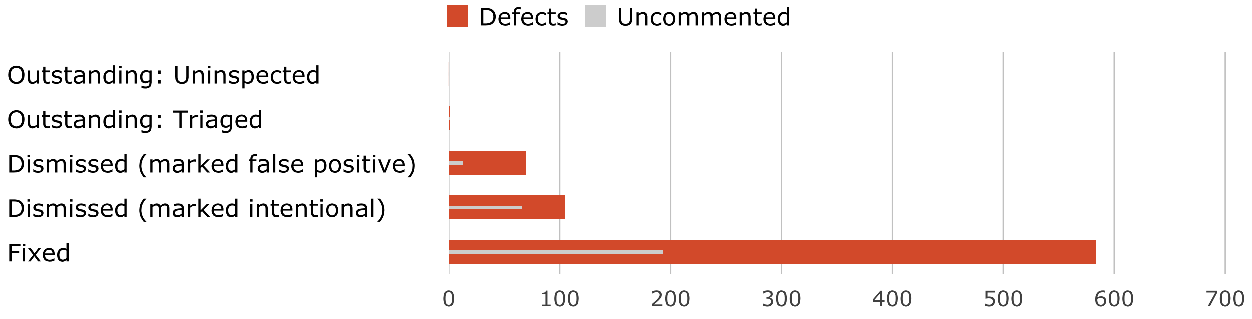


## Defect Severities by Component

| Component | Owner | Defect Density |
|---|---|---|
| Modules | | <0.01 |
| libffi | | 0.00 |
| ctypes | | 0.00 |
| Include | | 0.00 |
| Other | | 0.00 |
| Parser | | 0.00 |
| usr include | | 0.00 |
| libmpdec | | 0.00 |
| expat | | 0.00 |
| Objects | | 0.00 |
| Python | | 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.

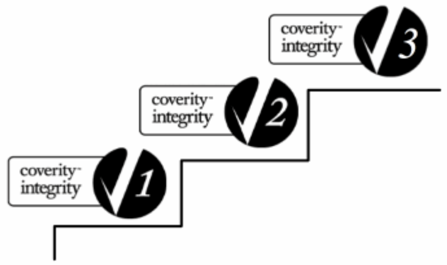
Coverity rating requirements are based on an assessment of several factors:

- 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.