BLACK DUCK®

WHITE PAPER

BUILD RELIABILITY AND SECURITY INTO YOUR SDLC

Defects and vulnerabilities become much more difficult and expensive to resolve when they're discovered late in the SDLC, or after the software has shipped. Code scanning tools that fit seamlessly into the developer workflow and CI/CD pipeline can uncover issues early—without impacting velocity.

Software is everywhere today, powering applications for almost every business while also providing critical capabilities in everyday products, such as automobiles, airplanes, and medical devices. But some types of coding defects can lead to catastrophic results, and security vulnerabilities put business systems and data at risk of being exploited. However, more than half of defects and vulnerabilities result from coding mistakes that could have been prevented. This has led many development teams to adopt coding standards and create security programs to help find issues earlier in the software development life cycle (SDLC), when they're easier to resolve.

It is no longer feasible to begin testing for defects and vulnerabilities during the later stages of the SDLC. Code scans must fit seamlessly into existing developer workflows and the CI/CD pipeline to enable the production of high-quality, secure software without creating delays.

ESSENTIAL ELEMENTS TO BUILD HIGH-QUALITY, SECURE SOFTWARE

To create high-quality secure software, these elements are essential.

- **A developer-centric approach.** Organizations must get developers' buy-in for any code testing tools if they are to be adopted successfully. These tools must support the programming languages, frameworks, and platforms developers use, and they must provide a high level of accuracy so that scan results don't become a distraction. Code scans should integrate directly into popular development tools, especially IDEs and code repositories, as well as CI/CD pipelines.
- **Comprehensive code analysis.** Some software defects can be easy to find, while others are difficult to identify, but both are critical to resolve before the software is shipped. Comprehensive analysis can uncover complex defects that span multiple files. As projects grow in size, the accuracy of this analysis becomes even more important, so developers don't waste their time triaging a large number of false positives.
- **Reporting and dashboards.** Stakeholders should be able to view key insights into software health, critical security vulnerabilities, developer productivity, and trend analysis to gauge progress toward business KPIs.
- **Compliance tracking and prioritization.** Defects and vulnerabilities should be mapped to the coding and compliance standards that matter to your business. This makes it easy to prioritize issues to drive compliance for a broad range of industry, functional safety, and security standards, including MISRA C/C++, CERT C/C++/Java, PCI DSS, and OWASP Top 10.

HOW COVERITY WORKS

Coverity® Static Analysis empowers developers and security teams to deliver secure, high-quality applications at scale. By building an in-depth model of each application; performing dataflow, control flow, and semantic analysis; and gathering insights into all dependencies, compilers, and more than 20 programming languages and 200 frameworks, Coverity can uncover complex issues that span multiple files and libraries across even the largest codebases. Potential defects are then validated by combining evidence about each issue with the context in which the code is used to help distinguish between real issues and false positives.

Critical issues that Coverity can identify include buffer overflows, memory safety errors, concurrency issues, race conditions, resource leaks, SQL injections, and cross-site scripting vulnerabilities. Coverity can also help identify policy violations based on industry, functional safety, and security standards as well as custom rulesets defined by users.

HOW COVERITY FITS INTO YOUR SDLC

Figure 1 shows a typical CI/CD pipeline. The workflow starts with developers writing code in their IDEs and then committing it to the repository. Coverity plugs into the IDE to identify defects as the code is being written, so issues can be resolved before they're committed. Scans can be automated to run on pull requests and integrated into popular code repositories and CI/CD tools, so issues are automatically displayed as comments right within key developer workflows.

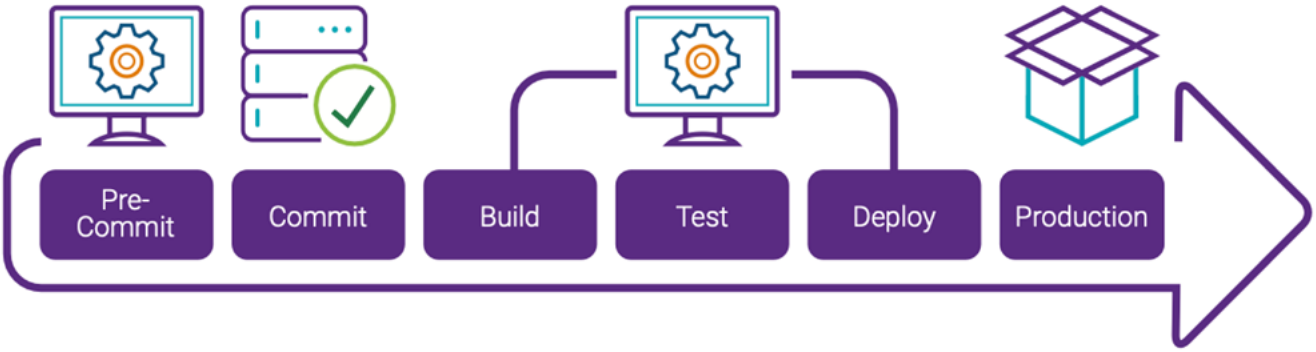

Figure 1. *A typical CI/CD pipeline*

Many organizations choose to run in-depth Coverity scans of their full project periodically. This provides a deep analysis of the entire codebase, which can uncover obscure, hard-to-find issues that span multiple files. Scan results can then be integrated into quality gates that can fail the build if critical defects or policy violations are discovered. Analysis can be configured according to the risk profile of the software and the organization's coding standards, ensuring that the analysis is tuned to produce highly accurate results.

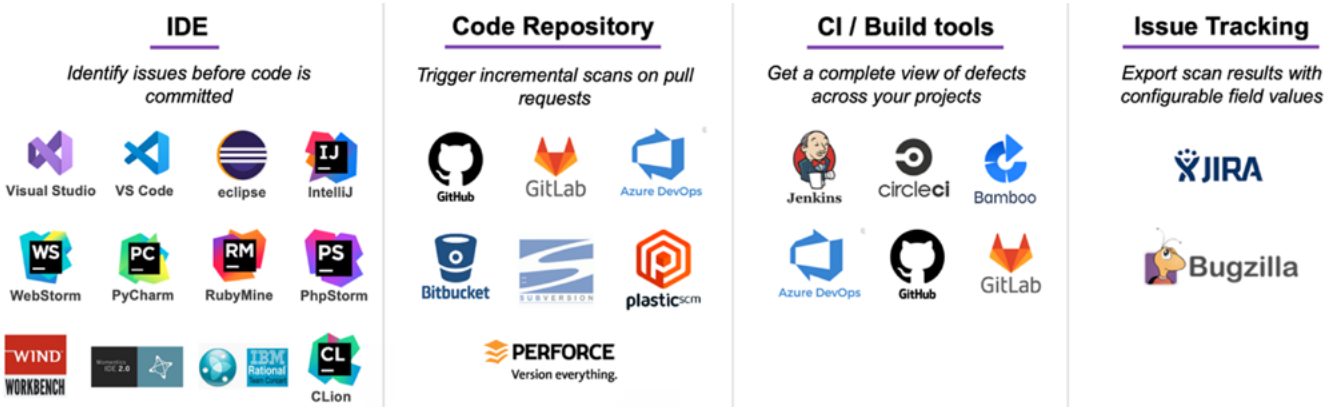

Figure 2. *Coverity SDLC integrations*

COVERITY COMPONENTS

The Coverity architecture provides organizations with the flexibility to deploy either on-premises or in a private cloud environment, and it can scale as needed to support even the largest software projects.

There are three key components of the Coverity architecture.

- Code Sight™ IDE Plug-in
- Analysis engine
- Issue management and reporting dashboard

Code Sight IDE Plug-in

Code Sight integrates with many popular IDEs such as Eclipse, as well as with less-common IDEs such as WindRiver Workbench. It identifies code quality and security issues using fast scanning techniques while developers write code, without requiring them to switch tools or manually invoke a scan. Detailed descriptions and actionable remediation guidance is included for each issue, enabling developers to produce high-quality secure code quickly.

Integration

Code Sight integrates with most commonly used IDEs, including IntelliJ IDEA, Visual Studio, and Eclipse. When Code Sight is installed, it detects Black Duck tools based on your product subscriptions and downloads the relevant engines in a matter of seconds. No further configuration is needed.

For static analysis, Code Sight downloads the Coverity analysis engine, which runs a high-fidelity file-level analysis behind the scenes every time a file is opened or saved. Developers can specify the scope of results, from one file to all open files. For each issue found, Code Sight provides a dataflow trace, including the line numbers of the main event and supporting events, to help developers understand the scope of the issue and find all problem areas in the code.

Code Sight automatically syncs with the Coverity server, whether deployed on-premises or in a private cloud environment, making the baseline for scan analysis easily accessible from the developer's desktop.

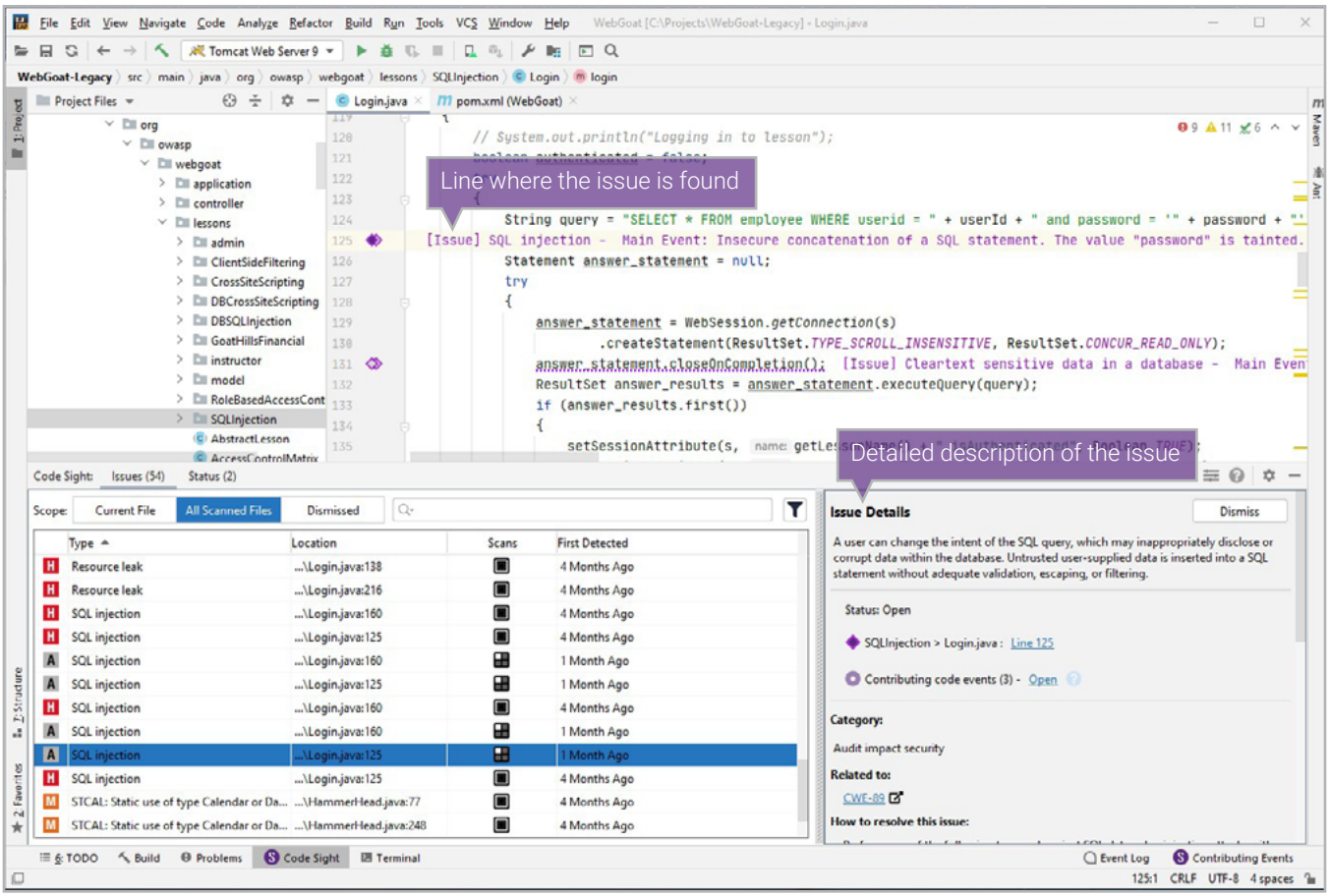

Figure 3. *Viewing issues found by Coverity in the developer's IDE through the Code Sight Plug-in*

Coverity analysis engine

The Coverity analysis engine is the horsepower running behind the scenes to find code quality and security defects in your software. Coverity analysis supports both build capture and buildless capture of issues, depending on the language. This enables integrations into both in-band development pipelines (via build capture) and out-of-band pipelines (via buildless capture).

Build capture

For compiled languages such as C/C++, Coverity supports build capture and wraps around a native build to analyze compiler operations and optimizations. This functionality helps the Coverity analysis engine gather information about dependencies and build-related programs.

Buildless capture

Buildless capture analysis requires minimal user knowledge and no build. It's particularly useful for security teams that don't have access to development or build tools. Users only need to provide the location of the project or repository. Buildless capture also allows users to limit the analysis to a specific set of files and exclude other files, such as test cases for library code, libraries the code doesn't specifically require, etc.

Issue management and reporting dashboard

After Coverity finishes analyzing project source code, users can view and manage defects and vulnerabilities in the Coverity Connect web portal, or in Code Sight. The Coverity Connect web portal lets developers, development managers, and security managers filter, triage, and prioritize issues according to the factors that matter most to their business. A wide range of prebuilt reports are provided, and users can create custom views to help prioritize issues, assess risks associated with applications, and present succinct findings to stakeholders.

Compliance reports

In addition to managing projects, many development and security managers need to generate compliance reports for a variety of industry, functional safety, and security standards (e.g., MISRA, CERT C/C++, OWASP Top 10). The Coverity Connect web portal provides highly intuitive reporting capabilities so stakeholders can accomplish their tasks. These reports help organizations create priority lists based on their risk assessment and focus on the issues that matter the most to them.

For example, organizations with functional safety requirements can filter and prioritize risks based on MISRA C/C++ or ISO 26262 requirements, and organizations developing web applications can prioritize based on risks relating to OWASP Top 10 categories. Both groups can use the Coverity Connect web portal to generate reports and assess and manage risks.

Trend reports

Each project has an issue trend report that is updated with each code scan, giving development and security managers insight into risks, progress toward achieving compliance, and productivity across teams.

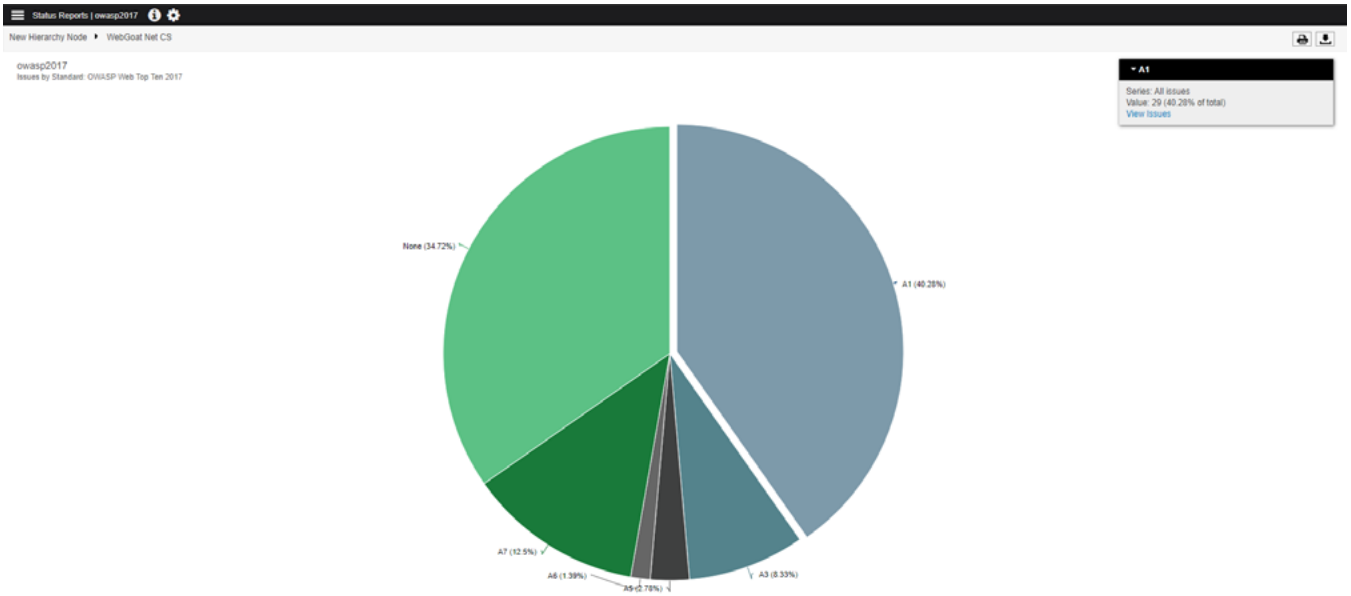

Figure 4. *An example of OWASP Top 10 category issues displayed as a pie chart in the Coverity Connect Policy Manager*

COVERITY LANGUAGE COVERAGE

Coverity supports more than 20 programming languages and 200 frameworks, including best-in-class coverage of C/C++ and Java projects, along with support for newer languages, such as Swift, Go, Kotlin, and Dart (see Figure 5). Language versions are updated on a regular basis to keep pace with advancements in these languages and frameworks.

Code scans identify a wide range of code quality and security defects across all languages, and map issues to industry, functional safety, and security standards. Code coverage includes multiple checkers for key categories of issues, such as those included in the OWASP Top 10. For example, Coverity runs more than 100 unique code checkers for the Java programming language to identify issues related to OWASP category A01 (Broken Access Control). Using the CWEs associated with Coverity checkers, organizations can also build their own internal coding standards and taxonomies.

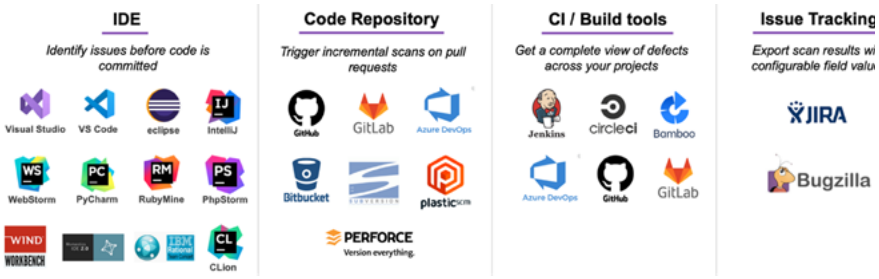

Figure 5. *Languages and frameworks supported by Coverity 2025.3.0*

FRAMEWORKS AND COMPILERS

Frameworks provide reusable software components as well as built-in capabilities that allow developers to build and maintain software applications more easily. Coverity offers deep support for more than 200 of these frameworks, providing it with the context to better evaluate software code, how it's being used, and the likelihood of potential defects ever being exploited. This leads to more-accurate analysis, including the ability to better distinguish between real issues and false positives.

When analyzing projects that use compiled languages, Coverity builds project source code and all dependencies into an intermediate representation of the software. This model is optimized to enable the analysis engine to efficiently identify defects and vulnerabilities, based on a detailed analysis of how the software is structured and operates. Because compilers use various approaches to evaluate program flows and capture source code, proper code analysis requires a clear understanding of each type of compiler. For some low-level languages with highly complex build environments such as C/C++, the make files and build system must be parsed to classify and capture code accurately. Coverity supports more than 50 compilers, which helps produce a highly accurate analysis of application flow and code usage. The Coverity documentation provides the complete list of supported compilers and frameworks.

VULNERABILITY IDENTIFICATION

Coverity identifies a wide range of code quality defects and security vulnerabilities, including

- API usage errors
- SQL injection
- Cross-site scripting
- Insecure data handling
- Hard-coded credentials
- Buffer overflows
- Cross-site resource forgery
- Memory safety errors
- Path manipulation
- Security misconfigurations
- Concurrent data access violations

Example 1: SQL injection

SQL injection is one of the most serious security vulnerabilities in software (see this list of examples). A SQL injection attack occurs when an attacker inserts tainted user data into a SQL statement that has a set of safety requirements or obligations. Figure 6 shows a snippet of a PhpMyAdmin application program block containing a possible SQL injection vulnerability. The orig_auth_plugin variable is assigned a user-supplied value that might include unsafe data, and concatenates it to a string that's intended to form a SQL query. If the application doesn't correctly sanitize this data, the tainted characters could change the statement into something unintended by the developer. Executing the statement could affect the confidentiality, integrity, and availability of the database system and enable the attacker to expose information that should be restricted.

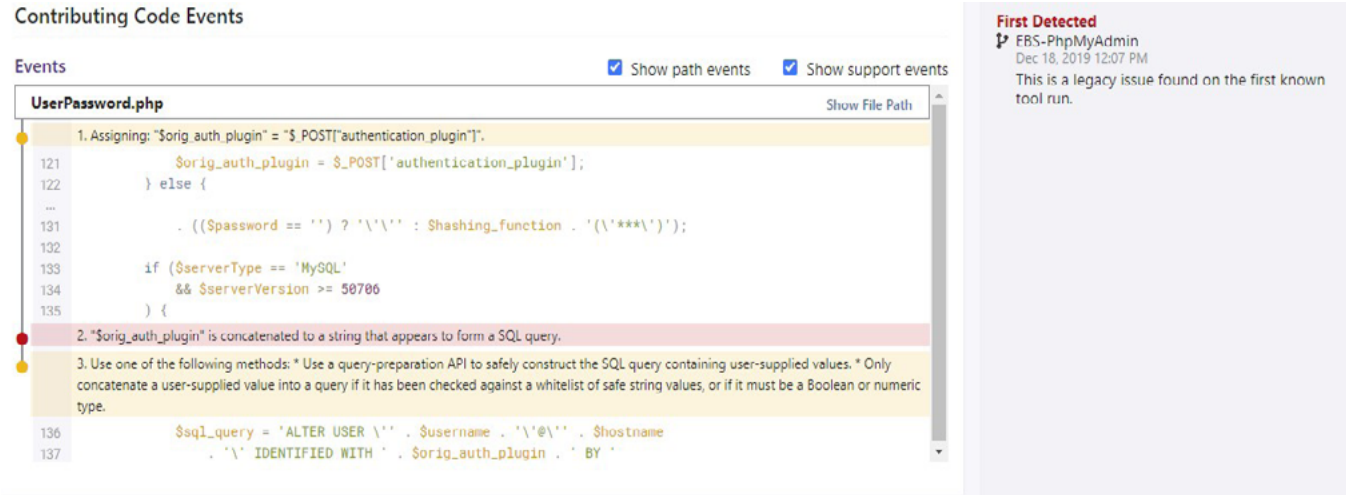

Figure 6. *PHP code snippet highlighting a possible SQL injection issue in a PhpMyAdmin application*

Coverity identifies critical issues such as SQL injection and offers remediation guidance. It also provides context-aware sanitizer libraries that identify the context related to unsafe data outputs, the appropriate sanitization methods, and the relevant technologies used in the code. Thus, Coverity can produce actionable remediation guidance based on the information it has about the program and help developers fix the issue quickly and accurately. For Java and C#, Coverity also offers an open source sanitizer library that includes sanitizers for difficult contexts not usually available in other libraries.

Example 2: Cross-site scripting

Identifying cross-site scripting (XSS) issues and providing context-aware remediation advice are also critical to shipping secure software quickly. According to the OWASP Top 10 list, XSS is the second-most-prevalent issue in web applications. This vulnerability allows an attacker to inject malicious, executable scripts into an application or website. Without proper sanitization, the script could be executed by the browser of an unsuspecting user visiting the application. Potential consequences of XSS attacks include an attacker stealing the user's keystrokes or cookie information, or redirecting the user to a malicious website.

To prevent XSS, an application must perform context-aware parsing and sanitization of HTTP requests. As explained in this blog post, input validation close to the data source can help protect against the exploitation of this vulnerability, but it will not remove the defect. Sanitizing the input—specifically, understanding how the data is being used to nest and escape HTML in the right order—near the data sink is necessary to remove XSS vulnerabilities.

Figures 7 through 9 show an example of an XSS vulnerability in a WebGoat-Legacy application, before remediation and after remediation by escaping the HTML. In Figure 7, the WebGoat-Legacy application shows an XSS vulnerability due to an unsanitized s.getWebgoatContext(). getFeedbackAddressHTML() function call. To eliminate this defect, the application must sanitize the input at the location where it's used (the sink). As shown in Figure 9, a developer can use Coverity's Escape library (imported, as shown in Figure 8) to escape the HTML and remedy the defect.

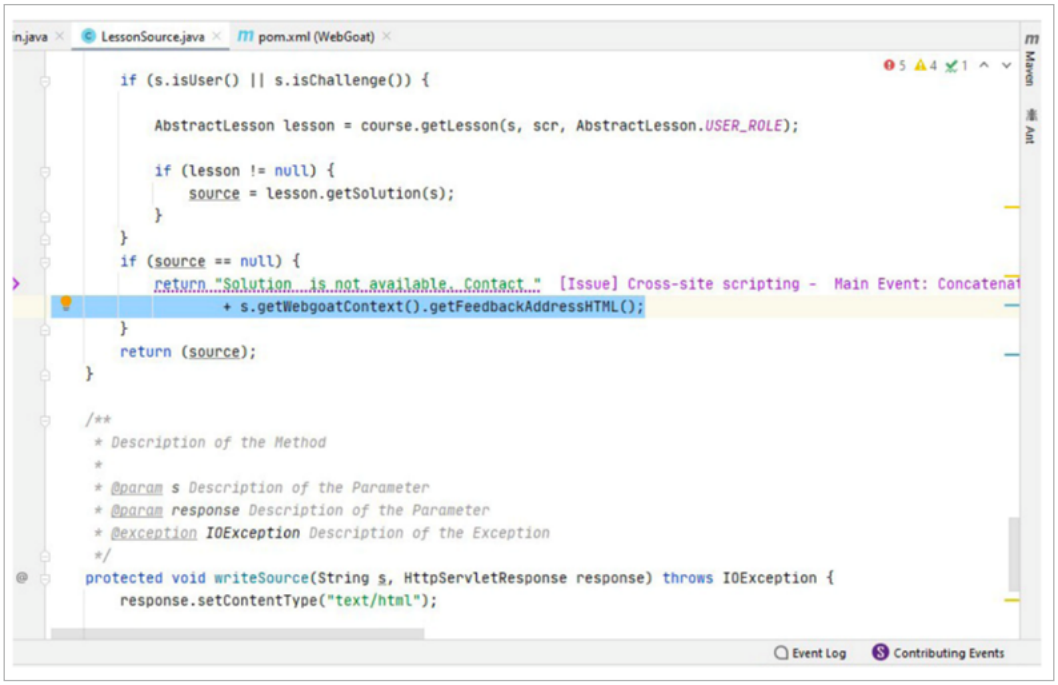

Figure 7. *The unsanitized s.WebgoatContext().getFeedbackAddressHTML() function call results in a cross-site scripting defect*

Figure 8. *The Escape library is imported in the source file*

Figure 9. *The Escape library imported from Coverity is used to escape the HTML and remedy the defect*

SUMMARY

Coverity provides developers with a comprehensive static analysis solution that offers in-depth code scans to uncover code quality and security issues, including complex issues that span multiple files and libraries. Automated scans can be triggered to run on code commits and pull requests to find issues early in the SDLC, without impacting release timelines. The Code Sight IDE Plug-in integrates Coverity scans into the developer environment to identify issues in new or changed code as the developer writes it, and before it's committed.

Coverity's support for a wide range of languages, frameworks, compilers, and integrations with developer workflows provides comprehensive analysis of any software project, even those with thousands of developers and millions of lines of code. Its high-fidelity scans, low-false-positive rate and fast analysis help development and security teams deliver reliable, high-quality, and secure software for any environment.

To learn more about Coverity or request a demo, please visit the Coverity static analysis web page.

ABOUT BLACK DUCK

Black Duck® meets the board-level risks of modern software with True Scale Application Security, ensuring uncompromised trust in software for the regulated, AI-powered world. Only Black Duck solutions free organizations from tradeoffs between speed, accuracy, and compliance at scale while eliminating security, regulatory, and licensing risks. Whether in the cloud or on premises, Black Duck is the only choice for securing mission-critical software everywhere code happens. With Black Duck, security leaders can make smarter decisions and unleash business innovation with confidence. Learn more at www.blackduck.com.