BLACK DUCK®

ENSURING RELIABILITY AND SECURITY IN C++ SOFTWARE

In the rapidly evolving landscape of software development, the use of open source components has become a cornerstone of modern application development. C++ projects in particular benefit significantly from the vast array of open source libraries and frameworks available. However, the integration of these components into enterprise software introduces a range of security and compliance challenges.

This eBook highlights the critical need to ensure that software written in C++ is free of defects and vulnerabilities. It also outlines key challenges and provides actionable solutions to address these issues. By leveraging the insights from the webinar "How to Score an A Plus on C and C++ in Open Source Security," this paper will help development leaders and organizations achieve a higher level of quality, security, and compliance in their C++ projects.

THE IMPORTANCE OF PRODUCING ROCK-SOLID C++ SOFTWARE

C++ is commonly used in the development of safety-critical systems, as well as in software projects across industries including automotive, IoT, and healthcare. Ensuring that these projects are secure and free of critical defects is essential for several reasons.

- **Personal safety:** In software that's involved with safety-critical systems, defects can lead to serious consequences including risks to personal safety, property, and the environment.
- **Cost of field updates**: Once the software is deployed in the field, fixing defects or vulnerabilities becomes very difficult and expensive.
- **Protecting sensitive data**: Some defects and vulnerabilities can expose sensitive data about your business or customers. This can impact customer trust, damage brand reputation, and lead to legal action.
- **User experience**: The success of commercial software is largely determined by the user experience. Defects and performance issues lead to a negative experience, which reduces customer satisfaction and erodes trust.
- **Regulatory compliance**: Many types of software must comply with industry, security, or government regulations such as MISRA, CERT C/C++, and others. Noncompliance can lead to product delays, loss of consumer confidence, and legal penalties.

KEY CHALLENGES IN ENSURING C++ SOFTWARE IS FREE OF CRITICAL DEFECTS

C++ is a powerful and versatile programming language that's widely used in various domains including system software, game development, and safety-critical components. The language's performance and flexibility make it an attractive choice for many developers. However, the complexity and low-level nature of C++ also introduce unique challenges. These challenges are exacerbated by the increasing reliance on open source components, which can introduce defects and vulnerabilities if not properly managed.

Key challenges to ensuring C++ software is free of critical defects and vulnerabilities include

- **Complexity**: C++ is a powerful and flexible language, but its complexity can lead to subtle and hard-to-find defects. Issues such as memory leaks, buffer overflows, and undefined behavior can be particularly challenging to detect and fix.
- **Outdated open source libraries:** Developers occasionally add components or code snippets from outdated or unmaintained libraries, and that can expose projects to known vulnerabilities.
- **Lack of package management:** The absence of a widely used package manager in C++ makes it challenging to automatically identify and track dependencies. This can lead to the inclusion of outdated or vulnerable libraries, increasing the risk of security breaches.
- **Scalability**: Many C++ projects have reached a massive scale, with some involving thousands of developers and tens of millions of lines of code. Testing tools must be able to handle the size and complexity of these systems, while maintaining accuracy.
- **Continuous integration**: Integrating security and quality testing into the continuous integration (CI) pipeline is crucial. This helps teams identify and fix issues early in the development life cycle, reducing the risk of critical defects in production.

STEPS TO ENSURE SECURE AND RELIABLE C++ SOFTWARE

Despite these challenges, many development and security teams have succeeded in delivering robust C++ software that their customers can count on. Some software may require additional measures, but there are steps that improve the reliability and security of even the largest and most complex C++ projects in the world. They include

1 Use both comprehensive static analysis and SCA scans

- **Finding**: Most static analysis solutions fail to accurately identify critical defects and vulnerabilities in complex and large-scale software projects. Additionally, traditional software composition analysis (SCA) tools fall short in identifying dependencies in C++ projects due to the lack of a standardized package manager and challenges in dependency resolution.
- **Best practice**: Integrate a comprehensive static analysis tool like Coverity® Static Analysis with an SCA tool like Black Duck® SCA. Coverity excels at identifying issues in very large and complex C++ software, including issues that span multiple files and libraries. Black Duck SCA accurately identifies both direct and indirect dependencies, even when they are linked in header files or external directories. When used together, they ensure that all your C++ code is free of defects and security vulnerabilities.

2 Create and maintain an up-to-date SBOM

- **Finding**: A comprehensive Software Bill of Materials (SBOM) provides real-time visibility into all components of your software to help identify risks before they can be exploited. However, software is constantly changing—new dependencies are added, old ones updated or removed, and new vulnerabilities are discovered and patched.
- **Best practice**: Use Black Duck SCA to create and manage your SBOMs on a continuous basis. This ensures that all open source and third-party components are accounted for, providing transparency and facilitating compliance with regulatory requirements, even as your software evolves over time.

3 Automate scans early in the SDLC

- **Finding**: In-depth code scans can be time-consuming and are often run later in the software development life cycle (SDLC), when resolving defects can impact release timelines. Managing open source components manually also requires a lot of time and can be error-prone. Automating scans to run earlier in the SDLC leads to higher-quality and secure code, without reducing development velocity.
- **Best practice**: Trigger fast static analysis and SCA scans to run before code is committed or on pull requests. This finds defects and vulnerabilities early, so they can be resolved before impacting other teams or release timelines.

AUTOMATING SCANS TO RUN AS PART OF EXISTING WORKFLOWS CAN

- **Detect issues when they're easiest to resolve**: Quickly identify defects and vulnerabilities in the code your developers write, and in open source and third-party components, without impacting velocity.
- **Identify dependencies**: Automatically detect and track all open source components, including those linked in header files and external directories, and those defined in Makefiles, CMake, or other custom build scripts.
- **Manage licensing risks**: Identify the licenses associated with open source components to ensure compliance with legal requirements.
- **Create robust SBOMs**: Generate comprehensive SBOMs that include all open source and third-party components, and export to standardized formats like SPDX and CycloneDX.
- **Improve operational efficiency**: Reduce the manual effort required to manage open source components, allowing development teams to focus on core development tasks.
- **Ensure up-to-date components**: Alert developers to newer versions of open source components, helping to keep the project up-to-date and reducing the risk of using outdated, potentially vulnerable libraries.
- **Facilitate regulatory compliance**: Track and prioritize issues based on the functional safety, security, and industry standards that matter most to your business.

The importance of developing C++ software that's secure and highly reliable cannot be overstated, especially when dealing with safety-related components or commercial software where business success is heavily affected by the user experience. Coverity and Black Duck SCA provide in-depth, accurate, and efficient analysis of C++ code to help eliminate critical defects and vulnerabilities, and deliver software on time that complies with the regulations that are most important to your business.

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.