Protecting the Embedded and IoT Software Build Environment **with Software Composition Analysis**

By IIoT World & Farallon Technology Group

Sponsored by

BlackBerry QNX

Table of Contents

Safely Navigating Software Quality and Security

Ensuring the quality, reliability and safety of software requires navigating a complex software supply chain made up of software engineers, operations managers, contractors, and independent software vendors (ISVs), along with open source software (OSS) providers. Oftentimes, device manufacturers that have outsourced software development or integrated third-party software are unable to examine the source code. This makes it difficult to verify that the software complies with safety standards and meets the manufacturer's own best practices for software development. Original equipment manufacturers (OEMs) rarely have access to all the source code used in their products.

Companies that have adopted agile software development methods have many software developers coding at the same time and submitting their code into a shared repository. Using a constant stream of small, incremental updates to the code base allows companies to integrate, build, test, and release software more quickly and with fewer defects. DevOps is a methodology that complements agile development by combining software engineering and operations to integrate and deliver high quality software faster and more reliably. DevOps uses processes and tools to streamline development and automate the orchestration and management of software delivery. Companies that develop their own software in-house have access to the source code and can more easily verify what goes into their applications; however, this does not mean that the software meets the company's requirements and other industry standards such as MISRA C 2012, ISO 26262 and IEC 62443 4-1.

OEMs that have licensed software to integrate into their products focus on ensuring that the software will meet their business and functional application requirements; however, licensing agreements rarely guarantee the delivery of non-functional requirements, such as security or secure software development practices, which can impact how the system should work.

> Software licensing agreements rarely cover non-functional requirements, such as security or secure software development practices

Protecting the software supply chain at each point within the software development lifecycle is critical to ensuring the quality and safety of software without slowing down the software development and delivery process.

Challenges of Hardening Software and Hardware

There are many challenges to hardening software and hardware in manufactured electronic products. “Hardening” refers to reducing the surfaces of attack and removing security vulnerabilities in the software, device, or system. Some of the primary challenges to hardening software, include:

- **Verifying non-functional requirements**
- **Reviewing compiled binaries**
- **Protecting the DevOps build environment**

Verifying Non-functional Requirements

Functional requirements define the basic behavior of the software application. Typically, functional requirements describe what the system should do under certain conditions. For example, a functional requirement could specify that when you press the button, the green light turns on.

Typical functional requirements include:

- Business rules
- Authentication
- Reporting requirements
- Certification requirements

Non-functional requirements describe how the system works in a way that impacts the user experience but that does not impact a functional requirement. For example, a functional requirement might specify that the system should authenticate the user based on a username and password. A non-functional requirement would be that the password should be a minimum length of 8 characters and include letters, numbers, and at least one symbol. Another non-functional requirement might be that the authentication method should verify the identity of the user using public key cryptography and digital certificates in addition to using a password. If you wanted to be specific, you could define another non-functional requirement in this way: “the public/private key pair used in PKI should be generated using a hardware security module (HSM).”

Typical non-functional requirements include:

- Performance
- Scalability
- Availability
- Security
- Manageability
- Data integrity

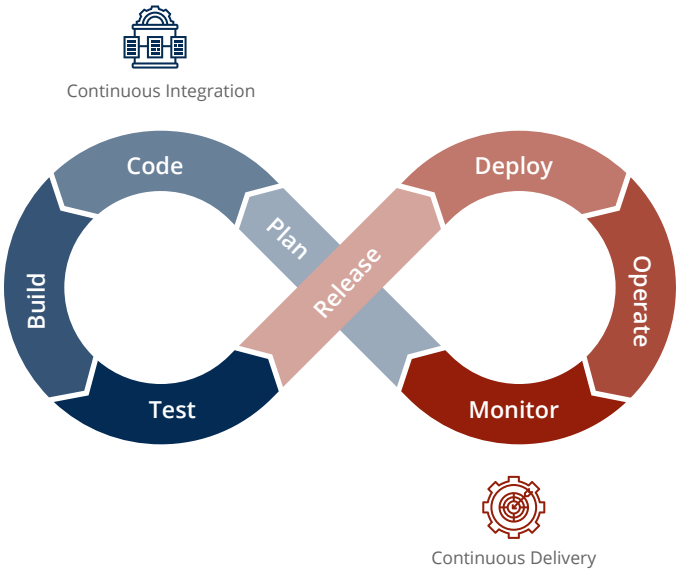
When designing a system to be secure, defining the non-functional requirements is as important, or more important, than defining the functional requirements because non-functional requirements can also introduce vulnerabilities. Too often, companies do not adequately define the non-functional requirements in sufficient detail when it comes to security and software development practices. Also, companies often eliminate non-functional requirements to reduce the scope of work (SoW) and cost of product development.

Verifying the functional and non-functional software security requirements for a product can be exceedingly difficult. If some of the software components are delivered as binaries, it could be difficult to test the non-functional requirements. Without testing the non-functional requirements, companies could fail to identify critical flaws and vulnerabilities in the software.

Reviewing Compiled Binaries

Reverse engineering compiled binaries back into source code is difficult and often violates the software licensing agreement. Binaries can often be decompiled or disassembled back into the component libraries that make up the application. This makes it easier to identify known vulnerabilities in the software sub-components and binary libraries that make up the application.

Protecting the DevOps Build Environment

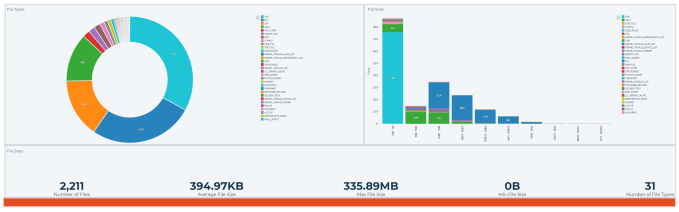


DevOps has embraced the practice of continuous integration and continuous deployment (CI/CD) for enterprise and cloud applications, whereby developers submit small, incremental code updates that are built, integrated, and tested automatically before being sent to the operations team to release into production. CI/CD enables companies to significantly reduce the software delivery time and increase the frequency of updates.

For embedded and IoT systems, product and software engineers are increasingly using DevOps methodologies to meet the demands for more frequent firmware updates and monetization strategies. As the software is being built and compiled, it is critical to constantly verify its safety, quality, and authenticity.

Software Bill of Materials (SBOM)

A software bill of materials (SBOM) is a list of components in a piece of software. Developers often create products that incorporate open source and commercial software with proprietary code within a single application or set of libraries. Understanding what is in the SBOM is an important first step in addressing the challenges of hardening software and hardware products.



Identifying Software Vulnerabilities, Paths and Dependencies

From a security perspective, software lives in two worlds. One world is the world of known risks, and the other is the world of the latent or yet-to-be-discovered risks. The world of known risks can be defined at any given point in time through careful software analysis, yet if unmanaged, will continue to grow to a potentially unmanageable level of risk. Moreover, the environment and configuration of the system and software can dramatically alter the level of risk exposure, both in a negative and positive way.

Let's begin by examining known vulnerabilities. As security researchers have continued to ply their trade over the last several decades, various organizations have taken up the noble project of compiling the known vulnerabilities into searchable databases. One of the most common and well respected of all vulnerability databases is the Common Vulnerability Enumerator (CVE) database originally compiled and managed by Mitre Corporation[1].

The CVE database forms the basis for the US NIST National Vulnerability Database (NVD),[2] which is regularly updated and can be accessed directly via their website or through web-based APIs, a common access method used by software composition analysis tools. The NVD is freely available and perhaps the largest and most comprehensive database of known software vulnerabilities globally.

Vulnerabilities are given a score based on a number of criteria to help determine the severity of the risk. While this can be a good starting point for determining how to prioritize risk, the score must be considered in a contextual manner in order to properly determine the true risk exposure of any given application or system. A software package, for example, may contain multiple vulnerable libraries that have been assigned high numerical severity scores, yet most of the libraries may not be exposed or accessible in the codebase. In this instance, the dependencies present at runtime will serve as a more accurate indicator of real risk. This is why it is very important to always combine the information gathered from the analysis software with expert human and contextual analysis.

Nonetheless, it can be successfully argued that once armed with the knowledge of what vulnerable software and hardware is present in the system and applications, you should do all that you can to eliminate all known vulnerabilities. This is important because it is extremely complicated to narrow down and prioritize risk when the software is bloated with a multitude of vulnerable libraries. Another consideration is that system configuration changes can dramatically alter the various runtime workflows and dependencies, and a once "secure" system or application can suddenly become high risk. Applying some thorough software and hardware hygiene and regularly re-examining the system and applications will serve to effectively manage security during the full lifecycle.

[1] Mitre Corporation Website. https://cve.mitre.org https://cve.mitre.org

[2] US NIST National Vulnerability Database. https://nvd.nist.gov https://nvd.nist.gov

Deconstructing the Software and Hardware Bill of Materials

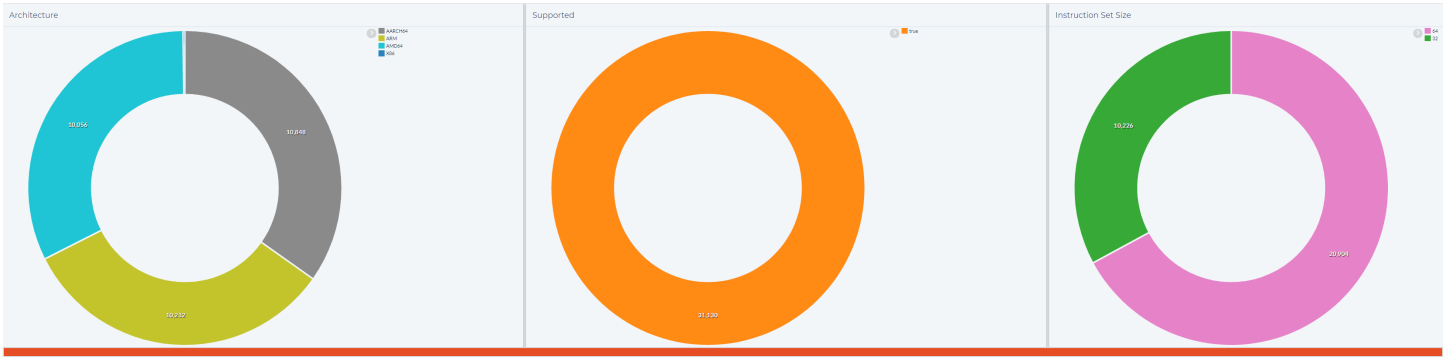
The world of both software and hardware development has evolved over the years. At one time both software and hardware were developed strictly from source code and painstakingly architected designs. As the world of software and hardware development evolved, it became readily apparent that building on previous work was far more efficient. Software developers now rely on previously compiled software libraries that can be obtained from online open source repositories, commercial development houses, or previously developed in-house libraries. This is also true for hardware developers that incorporate components into their devices and systems. Building on the successful work of others is now commonplace and far more efficient.

The challenge that arises is that both the internal development team creating the application and the end user (customer) have very little (or no) way of knowing the extent of any vulnerabilities that may exist both in the software and hardware. This is where a Software Bill of Materials (SBOM) and Hardware Bill of Materials (HBOM) become indispensable. Think of these as "ingredient lists" for software and hardware. By identifying the software and hardware "ingredients," a determination can be made about the potential vulnerability exposure for a given application. Once the software and hardware components are identified, they can be compared against known databases, such as the NIST National Vulnerability Database. Additionally, this information can be periodically or continuously cross-referenced with vulnerability databases to track vulnerabilities as they emerge over the lifecycle of the application, alerting both developers and users of risk exposure as it evolves.

Software Composition Analysis

Applications that are built using previously compiled libraries can be analyzed through a type of analysis known as software composition analysis (SCA). This type of analysis examines the compiled application and identifies the component libraries that were used to create it. It is especially useful because it does not require access to any source code. Binary analysis, a form of SCA, can be performed quickly and is non-destructive. While SCA can readily enumerate component libraries, typically it only lists the component libraries without making a determination of the security risks associated with the presence of such libraries. In some cases, SCA will discover that there are multiple versions of Java contained within a binary, and most may have significant security flaws, yet most will not be active in the compiled application.

It is important to select a SCA solution that can further analyze the application to determine dependencies and exposure during runtime. This can be partially covered by better analysis software; however, thorough analysis may require a much deeper view by someone trained in analyzing the results. Effective SCA software tools can automate much of this, yet it is best to look at analysis software that is well supported by services that can help make sense of what is discovered. An analogy is a person who goes to a lab to get blood drawn and analyzed. While the results may show that there are some causes for concern, a trained medical professional will review the results and take into account many more factors about the patient in order to determine if there is cause for immediate concern, or if it is simply important to make changes that can lead to a better analysis outcome.



Analysis of In-House Builds

SCA is an important tool for development organizations and in-house software development. A sizeable amount of the codebase created by software developers is based on building on previous code. While a developer may be familiar with what he or she is adding to the existing code, the previously developed codebase may not be as well vetted. This can be risky because latent security flaws will emerge over time as significant risks. As previously stated, SCA can be used to enumerate numerous active and inactive component libraries, and in some cases multiple versions that accumulate as "waste" as the codebase is modified over time. Proper code hygiene is essential to developing clean applications, and these unused "toxic" libraries should be purged from the codebase at every opportunity.

Analysis of Third-Party Components

Any organization that relies on third-party software and/or hardware components is constantly introducing unknown security risks into their environment. Some estimates indicate as much as 70 percent of code used by developers today comes from a third-party providing component libraries and approximately 90 percent of all code developed today relies on some level of third-party components during development. The risk does not stop with developers. Hardware manufacturers use third-party software applications for everything from toys to industrial controllers, as well as transportation, as motorized vehicles become more and more connected to networks and provide rich applications. Recently, some steps have been taken by governments to require an SBOM to be available for mission critical devices, such as medical devices and other industries are also considering this. SCA can readily provide this information to organizations looking to create an SBOM. Coupled with expert analysis, organizations can determine and manage their risks as they continue to utilize third-party components, as well as put pressure on third-party manufacturers to improve their security hygiene.

[3] Kirk, Jeremy. (2010, September 22). Third-party Code Putting Companies at Risk. Infoworld. https://www.infoworld/ https://www.infoworld.com/article/2626167/third-party-code-putting-companies-at-risk.html

[4] Wisseman, Stan. (2016, April 7). Third-party libraries are one of the most insecure parts of an application. TechBeacon. https://techbeacon.com/ https://techbeacon.com/security/third-party-libraries-are-one-most-insecure-parts-application

[5] House Sets Deadline for HHS to Develop Bill of Materials Action Plan. (2017, November 22). FDA News. https://www.fdanews.com/ https://www.fdanews.com/articles/12441-house-sets-deadline-for-hhs-to-develop-bill-of-materials-action-plan

Security for Hardware

It is important to understand that in order for a system or device to be secure, not only does the software need to be secure, but the hardware itself needs to be designed and managed to provide an environment for the applications to function with optimal security. Additionally, making a determination of the hardware build environment and components within a given system can be used to positively identify the context in which it operates. An example of this is identifying a hardware chipset that is known to have been developed specifically for automotive components. Identifying this chipset can help developers and security analysts prioritize security issues based on known threats, vulnerabilities, and compliance requirements specifically for the automotive industry.

In discussing secure hardware, there are several issues to consider. One issue is the instruction sets of the chipset used on the system or device. If secure operations are not fully supported, then the expected level of secure operations are compromised. Another consideration is the design and protective measures taken in building what is labeled as secure hardware, and specifically secure silicon. There are various certification programs that exist today that serve to deliver some level of assurance that the hardware on a system or device meets certain criteria. For example, both the US NIST Federal Information Processing Standard (FIPS)[6] and Common Criteria for Information Technology Security Evaluation (ISO/EIC 15408), referred to as Common Criteria or CC[7] have comprehensive requirements for multiple security assurance levels. However, it is important to understand that, despite having achieved requisite scrutiny for high assurance security certifications (e.g., FIPS 140-2 Level 3 or CC EAL Level 4+), security researchers and hackers have been known to find exploits that are effective against such hardware when given enough time and determination. This is why hardware composition analysis is so important. By analyzing embedded systems at a hardware composition level, it can be determined if requisite instruction sets are executing in a secure manner and what potential latent security issues may be present.

Final Product Build, Packaging, and Distribution

Software and hardware composition analysis should be performed on systems and devices during the final product build in order to determine if the system remains free of security vulnerabilities. The development stage can be quite long, and security issues can arise during the development time period that were not present in the earlier development stages. It is also important to understand that the way the software and hardware are configured can affect the security of the final build, since static builds do not tell the story of dynamic execution.

This is the stage where the product is ready to go to market. Product distributors and end-users are beginning to request a bill of materials to accompany delivered systems, as more of them become aware of the need to understand what they are implementing in their own networks and distributing to their customers. Beyond software and hardware composition analysis, it is also important to consider if the physical packaging used on the system is optimal for delivering a requisite security level. This can include anything from how the secure silicon is physically protected, to the presence or absence of IO ports either at a deep embedded level or at the network connectivity level (e.g., JTAG and USB). As the final product moves through the distribution network there is a risk of compromise at each checkpoint. It is important to have expert analysis determine what potential security risks may exist that may have been overlooked during the final build, packaging, and distribution.

[6] Federal Information Processing Standards Publications (FIPS PUBS). US NIST. https://www.nist.gov/

[7] Common Criteria Recognition Arrangement. Common Criteria for Information Technology Security Evaluation. Common Criteria Portal. https://www.commoncriteriaportal.org/

Standards Compliance

Developers and device designers look to industry standards to certify that their products meet requirements for safety and security. Software composition analysis platforms help developers and device designers to comply with industry standards.

MISRA C 2012

The Motor Industry Software Reliability Association (MISRA) C guidance is a set of software development guidelines for the C programming language developed by MISRA (Motor Industry Software Reliability Association). Its aims are to facilitate code safety, security, portability, and reliability in the context of embedded systems, specifically those systems programmed in ISO C / C90 / C99.

ISO 26262

ISO 26262, titled “Road vehicles – Functional safety”, is an international standard for functional safety of electrical and/or electronic systems in serial production road vehicles, defined by the International Organization for Standardization (ISO) in 2011, and revised in 2018.

IEC 62443 4-1

IEC 62443 4-1 specifies process requirements for the secure development of products used in industrial automation and control systems. It defines a secure development lifecycle (SDL) for the purpose of developing and maintaining secure products. This lifecycle includes security requirements definition, secure design, secure implementation (including coding guidelines), verification and validation, defect management, patch management and product end-of-life. These requirements can be applied to new or existing processes for developing, maintaining, and retiring hardware, software or firmware for new or existing products. The vulnerability testing requirements lists vulnerability scanning and binary file analysis as recommended types of testing.

CERT-C

The SEI CERT C Coding Standard is a software coding standard for the C programming language, developed by the CERT Coordination Center to improve the safety, reliability, and security of software systems.

Guidelines in the CERT C Secure Coding Standard are cross-referenced with several other standards including Common Weakness Enumeration (CWE) entries and MISRA.

CWE

CWE™ is a community-developed list of software and hardware weakness types. It serves as a common language, a measuring stick for security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.

ISO IEC TS 17961:2013

The purpose of ISO/IEC TS 17961 [ISO/IEC TS 17961:2013] is to establish a baseline set of requirements for analyzers, including static analysis tools and C language compilers, to be applied by vendors that wish to diagnose insecure code beyond the requirements of the language standard. All rules are meant to be enforceable by static analysis. The criterion for selecting these rules is that analyzers that implement these rules must be able to effectively discover secure coding errors without generating excessive false positives.

AUTOSAR C++14

AUTOSAR was developed in 2003 to address the explosion of automotive software, defining an open industry standard for electronic vehicle architectures. Over the years AUTOSAR has evolved to meet emerging automotive use cases such as autonomous driving and vehicle-to-everything (V2X) connectivity, resulting in the creation of the AUTOSAR Adaptive Platform.

UNECE WP.29

The UNECE World Forum for Harmonization of Vehicle Regulations (WP.29) is a worldwide regulatory forum within the institutional framework of the UNECE Inland Transport Committee. WP.29 defines the safety and environmental performance requirements for cars, vans, trucks, coaches, buses, powered two wheelers, agricultural vehicles, locomotives, and inland waterway vessels.

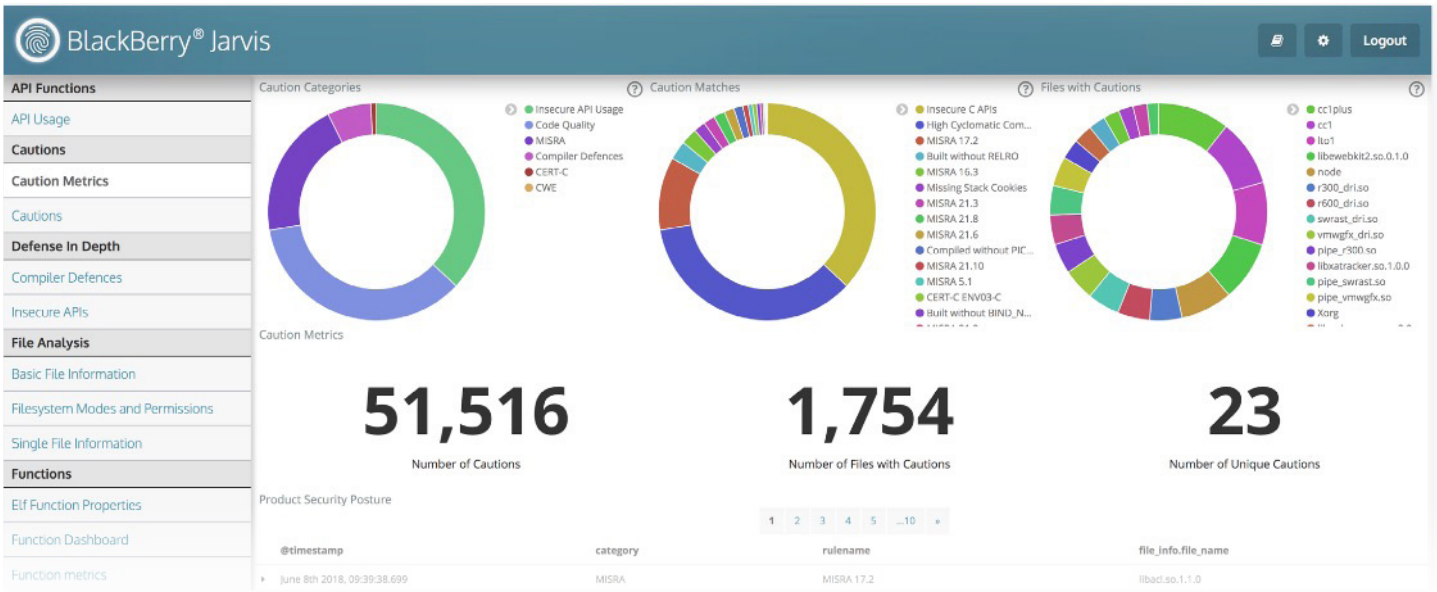
Summary and Recommendations

Ensuring the safety, quality and reliability of software requires navigating a complex software supply chain of developers, software engineers and third-party software providers. Original equipment manufacturers (OEMs) do not have the visibility into the software components that are integrated into their products. Rarely do OEMs have access to the source code for all the software in the software bill of materials (SBOM).

Hardening devices and software is difficult due to the challenge of verifying that both the proprietary and open source binaries do not contain known vulnerabilities. DevOps teams also want to make sure that the SBOM has been compiled in accordance with their own software development standards as well as industry standards for software development such as MISRA C 2012, ISO 26262, and CERT-C.

Identifying vulnerabilities and whether the application may be taking advantage of those vulnerabilities to compromise the system requires both an SCA as well expert services. Understanding the context with respect to the operating environment (chipset and operating system), industry, compliance standard, and build dependencies is critically important and is best accomplished with both tooling and security analysis.

BlackBerry® Jarvis™ – Software Composition Analysis for Embedded Systems and IoT



BlackBerry Jarvis is a cloud-based, software composition analysis platform that analyzes proprietary and open source binaries to identify security vulnerabilities without requiring access to source code. The platform inspects binary files in an easy, quick, scalable, and cost-efficient way, delivering deep insights into the quality and security of software components.

Through cutting-edge system exploration technology and a world-class, BlackBerry Jarvis provides powerful capabilities to examine a complete software product for security vulnerabilities and software craftsmanship.

Comprehensive Binary Analysis:
Identifies more than 100 categories of vulnerabilities in security and software craftsmanship

CI/CD Integration:
BlackBerry Jarvis is a SaaS-based tool with APIs for easy integration with CI/CD pipeline tools to analyze software at every stage of the DevOps lifecycle

Intuitive Dashboards:
Quickly identify vulnerabilities and prioritize response with powerful dashboards and CVSS scoring

Access to BlackBerry Best Practices:
Speed remediation with access to detailed descriptions and remediation advice based on BlackBerry's 20+ years of security experience

Contextual Analysis:
Improve vulnerability and compromise identification with contextual analysis of hardware, chipset, compliance, and code dependencies

Compliance Enablement:
Streamline compliance with industry standards with access to industry standards and customer-mandated assurance standards.

Since BlackBerry Jarvis extracts the characteristics and attributes from compiled binaries, access to source code is not required for organizations to gain insight into the final product. By building an SBOM and comparing the SBOM to known vulnerabilities databases, customer development requirements, and BlackBerry's own best practices, companies can understand what vulnerabilities exist and whether the software meets their functional and non-functional requirements.

BlackBerry Jarvis integrates with DevOps tools and CI/CD pipelines to automate binary analysis capabilities right into the workflow and build environment so that each build can be tested and verified before being sent to operations for deployment into production. BlackBerry Jarvis is able to identify more than 100 categories of vulnerabilities and uses context and discovery techniques to understand whether vulnerabilities are able to be exploited and whether there is evidence of an intent to exploit.

BlackBerry Jarvis enables companies to more easily meet the software development and safety standards for automotive and manufacturing. OEMs concerned about securing the software supply chain and managing their software build risks should consider implementing the BlackBerry Jarvis software composition analysis cloud-based platform.

About BlackBerry QNX

BlackBerry QNX is a leading supplier of safe, secure, and trusted operating systems, middleware, development tools, and engineering services for mission-critical embedded systems. BlackBerry QNX helps customers develop and deliver complex and connected next-generation systems on time. QNX technology is trusted in over 175 million vehicles and more than a hundred million embedded systems in medical, industrial automation, energy, and defense and aerospace markets. Founded in 1980, BlackBerry QNX is headquartered in Ottawa, Canada and was acquired by BlackBerry in 2010.

Learn more at blackberry.qnx.com

About IIoT World

IIoT World is a digital media outlet bringing the latest Industrial Internet of Things (IIoT) virtual conferences and content to a global community of 200,000 decision makers and influencers. IIoT World focuses on delivering daily intelligence on the IIoT, artificial intelligence, augmented reality, predictive analytics, digital disruption, autonomous cars, cyber security, machine learning and smart cities. Online, IIoT World curates a series of virtual events that draw thousands of delegates, including IIoT World Days, the largest Industrial IoT virtual event in the world. www.iiot-world.com

About Farallon Technology Group

Farallon Technology Group is a technology research and advisory firm focused on embedded, operational technology (OT), and cloud native cybersecurity. We deliver technical market and vendor research that covers key security technologies for cloud, IoT and edge computing. Farallon helps industrial companies, OEMs, and cybersecurity vendors to navigate business and technology risks and opportunities to reduce cost, drive digital transformation, and manage supply chains. www.farallontech.com