

GUIDE TO SECURING YOUR SOFTWARE SUPPLY CHAIN

Before It's Too Late

ONEKEY

WHITEPAPER

SUMMARY

Guide to Securing Your Software Supply Chain

Reading time: Approximate reading time: 15 min.

This whitepaper is designed for C-level leaders (CIOs, CTOs, CISOs), supply chain risk managers, and security and compliance teams tasked with protecting their organizations against hidden software supply chain risks, including emerging AI-driven risk. It also supports firmware engineers and technical practitioners with practical guidance on binary-level visibility, SBOM validation, and vulnerability management. It is relevant for industries where firmware and embedded systems are critical, such as telecommunications, automotive, medical devices, manufacturing, and critical infrastructure.

Software supply chain security is now a boardroom concern. As more embedded systems rely on third-party components, firmware, and internal development, the chances of hidden vulnerabilities and compliance issues continue to rise. Today's supply chains are complex and lack full transparency, which makes it easier for serious risks to go unnoticed.

This whitepaper outlines how the reality of modern development often skips or overlooks key security checkpoints. Tight deadlines, opaque firmware, and fragmented tools make it challenging to validate firmware before release or monitor it after deployment. Meanwhile, regulations like the EU Cyber Resilience Act, U.S. Executive Order 14028, NIS2, etc., are setting clear expectations for security, traceability, and accountability across digital products.

According to Gartner, “by 2025, 45% of organizations will have experienced software supply chain attacks — a threefold increase from 2021.”¹ In response, businesses are being pushed to shift left, embedding security earlier in development and not waiting until after products or firmware ship. That's where automation makes the difference. Integrating automated tools into the firmware development process enables organizations to analyze binaries without requiring source code, detect vulnerabilities earlier, enforce policies, monitor deployed devices, and ensure compliance. Automation turns best practices into repeatable actions, reducing operational risk and cost. To illustrate the business impact of this automated approach, this whitepaper examines how Swisscom implemented ONEKEY's automated solution, enabling them to save 400K USD per incident avoided. If securing your supply chain feels out of reach, this whitepaper shows you how to make it part of how you build and lead.

¹ <https://www.gartner.com/en/newsroom/press-releases/2022-03-07-gartner-identifies-top-security-and-risk-management-trends-for-2022>

INTRODUCTION TO SOFTWARE SUPPLY CHAIN SECURITY



Supply chain refers to the movement of products from development to deployment. In the digital world, we call it “Software Supply Chain,” and no doubt, everyone is aware of its complexity. The software supply chain is a series of steps, tools, and components that transform code into the software or firmware that ultimately runs on smart devices and embedded systems.

Just as a car is built from thousands of physical parts, sourced from multiple suppliers, software today is rarely created from scratch. It is assembled from:

- Codes written by internal teams
- Open-source software libraries and frameworks
- Commercial 3rd party components
- AI-Generated Code
- Supplier-delivered binaries and firmware modules

These components are integrated, transformed, and packaged into the software or firmware that is deployed on modern devices.

Returning to the car manufacturing example, rigorous quality control, compliance checks, and traceability are essential across the supply chain to ensure that each component meets safety and performance standards.

Similar to it, there is “Software Supply Chain Security” in the digital world. Simply, it refers to practices, processes, and technologies used to ensure that all code, components, and libraries involved in building, producing, and distributing software or firmware are trustworthy, secure, and compliant.

As this whitepaper focuses on supply chain security in the embedded world, where devices are built for specific functions with unique operational requirements, the discussion will center on the firmware that enables these functions. In some sections, you may see the term ‘software’ used more broadly. This is intentional and reflects the overlap in concepts.

Key Roles in Software Supply Chain

Understanding who is responsible for what in the software supply chain is critical to building an effective security posture. In the software supply chain, there are various roles such as asset owners, maintenance service providers, integration service providers, and product suppliers. For simplicity of this whitepaper, we will consider two categories such as vendor (product suppliers) and operator (asset owners, maintenance service providers, and integration service providers).

Vendor/Developer:

This group mainly focuses on developing firmware, assembling components, supporting, and delivering finished products to customers, operators, or downstream partners.

In the ideal world, their prime responsibility would be to ensure the integrity, security, and compliance of what they build as well as deliver it to their clients.

Operator:

This group focuses on managing and operating assets, as well as maintaining, commissioning, and validating the deployed system. The operator's assets might include enterprise systems such as virtual machines, network equipment, etc. Examples of embedded and IoT systems are industrial controllers, automotive ECUs, robotics, drones, etc.

In the ideal world, their prime responsibility would be to monitor and manage the firmware security running across their environments and assets.



Software Supply Chain from Vendor/Developer to Operator

In an ideal scenario, the software supply chain consists of approximately five stages spanning development to deployment and is considered secure, trustworthy, and verifiable. It integrates security validation, compliance enforcement, and continuous monitoring. The following diagram illustrates the ideal software supply chain along with critical security components.

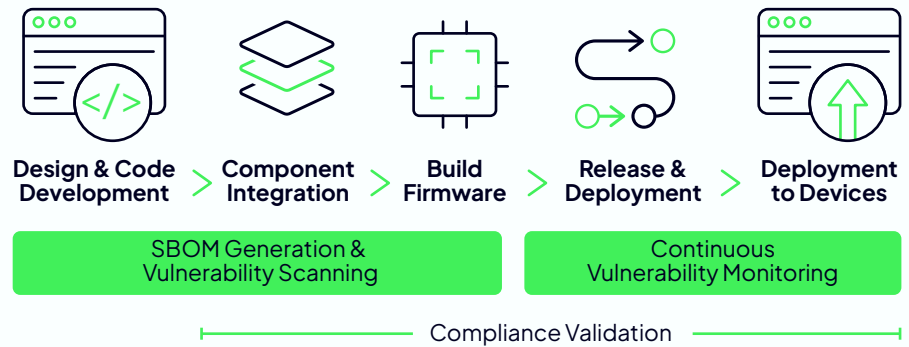


Figure 1: An ideal software supply chain integrates security validation during development and build, enforces compliance before release, and maintains continuous visibility and control during operational use.

The five stages of the ideal software supply chain, along with security measures, consist of:

1. Design and Code Development

Before any line of code is written, the process begins with secure-by-design thinking, incorporating security and compliance requirements during the design phase itself. This ensures that architectural decisions, threat models, and component selections all align with your organization's risk posture and regulatory needs. Following this, developers begin writing proprietary code and implementing functionalities based on user stories and functional requirements. They also add the relevant dependencies.

Best Security Practice: From the beginning, organizations should define security and compliance policies for all components, both internal and external, based on the principles established in the design phase.

2. Component Integration

Developers integrate open-source packages, third-party libraries, and supplier-delivered binaries, as modern firmware is rarely built from scratch to implement common functionality without the need to develop everything in-house.

Best Security Practice: Teams should generate a Software Bill of Materials (SBOM) and perform vulnerability scanning, ensuring that all components are known, scanned, approved, and if they are still actively maintained to fix newly discovered vulnerabilities.

3. Build Firmware

At this stage, code, integrated components, and their dependencies are compiled and packaged into firmware.

Best Security Practice: Before any release, security and compliance quality gates must be enforced to confirm that all components pass defined criteria and work together as expected.

4. Release and Deployment

Once approved, the firmware is released either to staging, test, or production environments. At this point, deployment readiness is verified.

Best Security Practice: Firmware should be accompanied by compliance documentation and an SBOM, enabling downstream teams to verify what's inside.

5. Deployment to Devices

Operators install firmware across environments ranging from cloud and enterprise systems to embedded devices and industrial endpoints.

Best Security Practice: This stage must include ongoing monitoring, tracking vulnerabilities over time, detecting regressions, and managing the patch/update cycle.



Layers of Firmware Complexity

In an embedded system, firmware sits on top of hardware and spans several distinct layers. The application layer is where the complexity of the software supply chain truly begins. This is the layer where the main product functionality is implemented, often blending internally developed code, third-party libraries, and supplier-delivered components, each with its own dependencies. Every piece introduced here adds potential risk and dependency chains that ripple through the rest of the system.

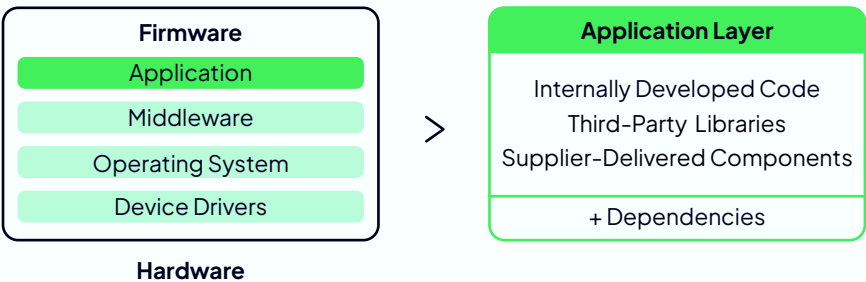


Figure 2: Layers of Firmware Complexity in Embedded Systems

Other layers are summarized as follows:

Middleware: The middleware bridges the application and lower layers, making it easier for different parts of the embedded system to work together. It often includes device drivers, communication protocols, and security services. **Operating System:** It manages system resources like memory, CPU, and peripherals, ensuring efficient operation. **Device Driver Layer:** This layer directly controls hardware components with device drivers such as sensor drivers, actuator drivers, and peripheral drivers. **Hardware Layer:** It is the foundation of the system, which includes CPU, memory, and peripherals like sensors, actuators, communication interfaces, timers, etc.

The software supply chain is evolving rapidly to meet the demands of faster development and delivery. At the same time, its structure is becoming more layered and complex, involving numerous tools, components, and platforms. This complexity brings a growing challenge of understanding and controlling what’s actually being assembled and deployed. As a result, risks and vulnerabilities across the supply chain are steadily increasing.

REALITY OF SOFTWARE SUPPLY CHAIN AND ASSOCIATED RISKS

In an ideal scenario, the software supply chain should include rigorous validation before release and continuous monitoring after deployment. However, many organizations struggle to implement these steps consistently.

Reality of Software Supply Chain

Many software supply chains skip or only partially perform security and compliance checks. Pressures like fast release cycles, limited supplier visibility, and fragmented tooling often lead to incomplete SBOMs, unchecked third-party components, and shallow vulnerability scans. Post-deployment monitoring is frequently missing, especially in embedded systems. As a result, hidden risks can move unchecked into deployment.

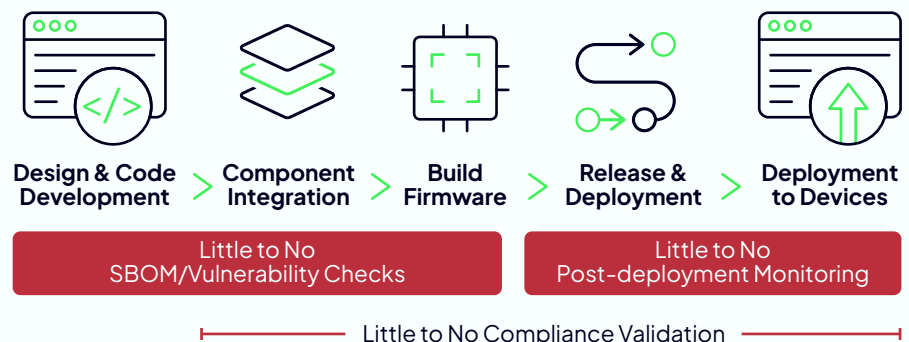


Figure 3: In many organizations today, the software supply chain prioritizes delivery speed but lacks consistent security validation and post-deployment monitoring, allowing risks to propagate downstream silently.

Reality of Security and Compliance Validation (Pre-release)

- Many organizations do partial or incomplete validation
- Some vendors skip the validation entirely, especially if they are under time-to-market pressure or if their systems are not well integrated with security tools
- SBOMs are often incomplete or missing
- Vulnerability scans may or may not be done with traditional scanners. If done, the organizations don't know which vulnerabilities are relevant



- Third-party and supplier binaries are often treated as black boxes; no validation is performed on them at all

Consequences: Vulnerabilities, malware, or non-compliant components may be shipped, creating gaps in the security of the supply chain.

The reality of missing or incomplete SBOMs and compliance validation (especially for CRA) are backed up by our latest IoT & OT Cybersecurity Report 2025. It states that creating SBOMs and compliance with principles of “Secure by Design” and “Secure by Default” are in top 3 challenges for many organizations. Link: <https://www.onekey.com/resource/iot-ot-cybersecurity-report-2025>

Reality of Operational Monitoring (Post-deployment)

- Many organizations lack mature monitoring of deployed firmware
- Sometimes, there might be no process for re-scanning SBOMs after initial deployment

Consequences: Vulnerabilities in deployed firmware remain unpatched for an extended period, and attackers take advantage of the long attack window to exploit vulnerabilities

Real-World Impact: SolarWinds Supply Chain Attack²

The SolarWinds attack is one of the most well-known examples of a software supply chain breach. In this incident, attackers compromised the company’s build environment and inserted a malicious backdoor into a routine software update. That compromised update was then unknowingly distributed to over 18,000 customers, including major enterprises and U.S. government agencies. The breach highlighted how a single point of failure in the software supply chain can have a massive downstream impact, reaching even the most secure environments.

Takeaway: Without deep visibility into the final firmware binary, malicious code can pass through trusted channels undetected and propagate at scale.

Real-World Impact: XZ Backdoor Discovery Reveals Linux Supply Chain Attack³

In early 2024, a critical backdoor was discovered in the popular open source XZ Utils compression library, used widely in Linux distributions. The attacker had patiently gained the trust of the open source maintainers over the years, eventually slipping malicious code into the project’s updates. Once deployed, this backdoor allowed remote access to affected systems. The incident demonstrated how even core, long trusted open source components can be compromised at the source, impacting countless downstream products and services.

Takeaway: Even widely used and “trusted” components can be weaponized upstream, making binary level inspection essential before integration.

² <https://www.techtarget.com/whatis/feature/SolarWinds-hack-explained-Everything-you-need-to-know>

³ <https://www.techtarget.com/searchsecurity/news/366577602/XZ-backdoor-discovery-reveals-Linux-supply-chain-attack>

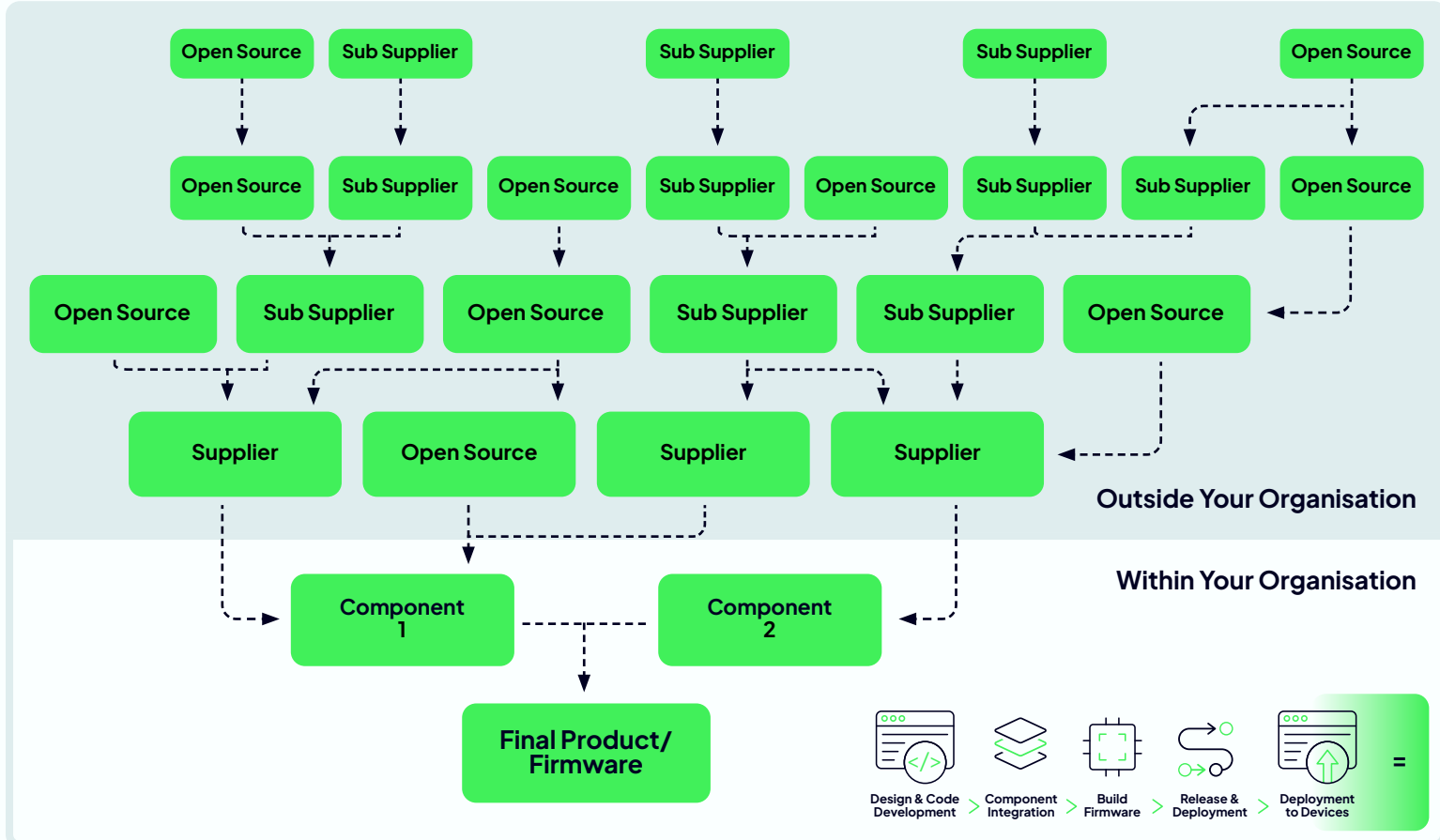


Figure 4: The software supply chain is a complex ecosystem in itself due to the number of dependencies, and can be called the Embedded Software supply Chain

Complex Web of Software Supply Chain

The software supply chain is rarely linear. As shown in Figure 4, it functions as a complex web of interconnected vendor (your final product/firmware), suppliers, sub-suppliers, internal teams, and open-source each contributing components, code, or processes at different stages. Before firmware ever reaches its destination, it typically passes through multiple rounds of code development, component integration, building firmware, and deployment into testing, often across different systems, teams, and even organizations. This layered network of dependencies makes the supply chain highly scalable but also highlights just how intricate and distributed the path to deployment has become.

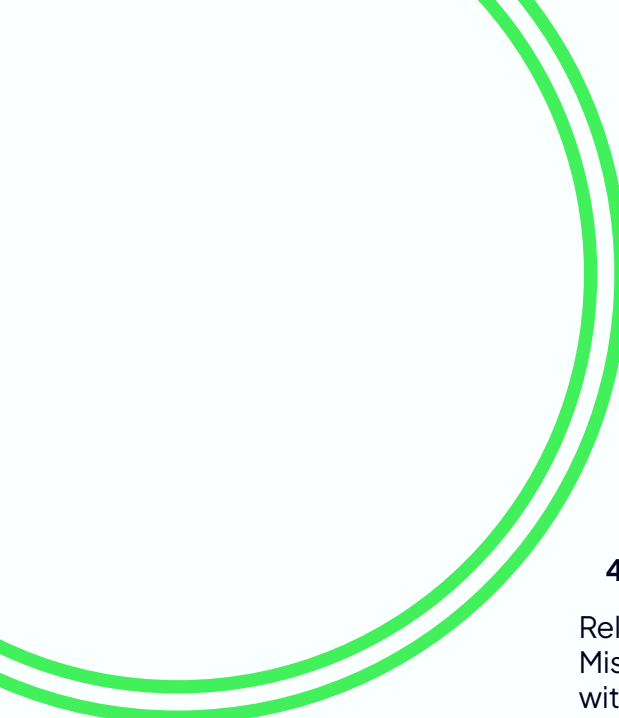
Associated Security Risks

The software supply chain isn't just long, it's also tangled. Due to the missing or unavailable security and compliance checkpoints at different stages, the supply chain is always at risk.

Some of the associated risks with each stage include:

1. Design and Code Development

Missing security-by-design principles lead to architectural shortcuts, absent threat models, or ignored secure patterns. This not only creates structural weaknesses but also allows insecure code and unvetted libraries to enter early, making issues harder and costlier to fix later.



2. Component Integration

Third-party libraries and supplier binaries are commonly added without deep review. These components may carry transitive vulnerabilities buried in nested dependencies that aren't visible.

3. Build Firmware

Without proper checks, build-time injections, configuration errors, and image vulnerabilities can be introduced during packaging. Incomplete SBOMs and weak validation leave teams guessing what's actually in the final firmware.

4. Release and Deployment

Releases often move forward without verifying what's inside. Misconfigurations and insecure settings can slip through, along with unverified build output and compliance gaps that weren't caught earlier.

5. Deployment to Devices

Once firmware is deployed, periodical scanning for vulnerabilities is rarely performed. Without visibility, known vulnerabilities go unpatched, and zero-day vulnerabilities can sit undetected, especially when there's no process to track what's actually running in the field.

Emerging Supply Chain Risks from AI-Coding

AI coding assistants like ChatGPT, GitHub Copilot, CodeLlama, etc., are rapidly being adopted by developers for firmware development to boost productivity. However, these AI coding tools introduce new risks to the software supply chain by suggesting incorrect or nonexistent libraries and components.

1. Slopsquatting (Exploitation of AI Mistakes)

AI coding assistants are trained on massive datasets that include public code repositories, documentation, and other sources from the internet. The models, for example, ChatGPT, GitHub Copilot, or CodeLlama, don't understand software the way humans do, as they are still in initial phases of their lifecycle. They predict the next-best-looking piece of code based on the patterns they've seen in training. In doing so, they might hallucinate a package name that sounds believable and suggests nonexistent or mistyped dependencies. Cybercriminals and malicious actors monitor the popular LLM tools and analyze hallucinated or likely-to-be-suggested names. They then pre-register those fake packages (known as slopsquatting). Once the package is in the registry, any developer who blindly trusts and installs AI-suggested code becomes vulnerable.

2. Vibe Coding Fuels Slopsquatting Attack

Vibe coding refers to a fast-paced, intuition-driven development style where developers prioritize speed and creativity over rigorous validation. In this type of workflow, the code is often written without peer reviews, dependencies are added based on what feels right, and AI tools are blindly trusted to fill in the blanks.

Consequences: Vibe coding creates the perfect storm for slopsquatting. When developers skip validation in favor of speed, they unknowingly rely on AI-generated hallucinations. If attackers anticipate these, they can inject malicious packages at scale. That makes even early-stage, low-risk projects a potential entry point into your software supply chain.

As these risks grow in scale and consequence, regulatory bodies worldwide are stepping in. From the EU Cyber Resilience Act to U.S. Executive Orders and FDA requirements, the message is clear: “Your organizations must demonstrate transparency, traceability, and accountability across their software supply chains.”



COMPLIANCE REQUIREMENTS FOR SOFTWARE SUPPLY CHAIN SECURITY

Regulators and governments are taking a more active role in defining how your organization secures its software supply chains. They are stepping in and setting the terms. What was once considered a “nice to have” is now becoming a mandatory requirement. Across industries and regions, new laws and frameworks are raising the bar on transparency, vulnerability management, secure development, and supplier accountability. This section highlights the most relevant and partially relevant compliance mandates that are shaping how organizations approach supply chain security.

Relevant Regulations and Standards

These regulations directly address the security of the software supply chain, including SBOM, vulnerability management, monitoring, etc. If an organization is building connected devices and firmware-based products in a regulated industry, these regulations and standards are coming into play.



Regulation / Authority	Region	Vendor/ Developer or Operator	Mandatory or Voluntary	Manu- facturing	Auto- motive	Medical	Tele- commu- nication	Critical Infra- structure	Description
EU Cyber Resilience Act (CRA)	EU/Global	Both	Mandato- ry (once enforced)	✓			✓	✓	Mandates security features, vul- nerability handling, and SBOMs across digital products.
Radio Equipment Directive (REDII)	EU/Global	Both	Mandatory	✓			✓	✓	Cybersecurity requirements for internet-connected radio equipment devices; mandatory by Aug 2025.
NIS2 Di- rective	EU	Operators Primarily	Mandatory	✓	✓	✓	✓	✓	Requires risk management, supplier controls, and incident reporting. Reduce operational risk by actively scanning for vul- nerabilities in assets or devices. It is relevant for organizations that fall under the NIS2 Directive.
UNECE WP.29/ R155	EU	Both	Mandatory		✓				Automotive regulation focused on cybersecurity management systems and continuous risk management.
Executive Order 14028	USA	Vendors/ Developers Primarily	Mandatory	✓	✓	✓	✓	✓	Promotes SBOMs, secure soft- ware development, and incident response; primarily aimed at federal agencies, but applicable broadly.
FDA Cyber- security Guidance	USA	Vendor/ Developer	Mandatory			✓			Requires SBOMs, threat models, and patchability for medical devices.
NIST 800- 218 (SSDF)	USA	Both	Voluntary	✓	✓	✓	✓	✓	Framework for secure software development practices.
METI Cyber- security Guidelines	Japan/ Global	Both	Voluntary	✓	✓	✓	✓	✓	Covers SBOM use, supplier soft- ware risk, and secure develop- ment practices for embedded systems.
CSA CII Supply Chain Progra	Singapore	Both	Mandatory	✓	✓	✓	✓	✓	Outlines software supplier risk controls, secure development, and SBOM practices for critical infrastructure
ISO/SAE 21434	Global	Both	Not Direct- ly Manda- tory - But Required for UNECE R155		✓				Cybersecurity engineering stan- dard for road vehicles.
IEC 62443	Global	Both	Voluntary	✓				✓	Industrial and OT cybersecurity framework applicable to em- bedded devices.
IMDRF Cyber- security Guidance	Global	Both	Voluntary			✓			Defines SBOM practices for medical devices, covering supplier software, transitive dependencies, and lifecycle risk management.

Partially Relevant Regulations and Standards

These mandates may not explicitly call out the software supply chain, but they touch areas where supply chain security plays a critical role, such as device safety, secure sourcing, or update integrity. They're still worth paying close attention to, especially for organizations operating across borders or in risk-sensitive sectors.

Regulation / Authority	Region	Vendor/ Developer or Operator	Mandatory or Voluntary	Manu- facturing	Auto- motive	Medical	Tele- commu- nication	Critical Infra- structure	Description
EU MDR (Medical Device Regula- tion)	EU	Vendor/ Developer	Mandatory			✓			Requires software validation and verification if medical devices are shipped with embedded software.
ETSI EN 303 645 (IoT)	EU/Global	Vendor/ Developer	Voluntary	✓	✓		✓		Baseline cybersecurity stan- dard for consumer IoT devices, including updates and access controls.
Protective Security Policy Framework (PSPF)	Australia/ Global	Both	Mandatory				✓	✓	Primarily addresses general sup- ply chain and procurement risks.

Disclaimer: The regulations and standards listed in the above table represent a few examples that are partially relevant to the security of the software supply chain according to ONEKEY. They may be relevant to your organization based on your unique security posture and operational context.

The message from regulators is getting harder to ignore; organizations are expected to know what's in their firmware, where it came from, and how it's being managed. It's not enough to react after something goes wrong. Fixing issues post-release isn't the goal; avoiding them in the first place is. That means shifting security left, and building it into development workflows instead of treating it as a last-minute add-on.

SHIFT SECURITY LEFT TO STOP BREACHES EARLY

Modern embedded systems rely on complex supply chains filled with third-party components, open-source libraries, vendor-supplied binaries, and AI-generated code. The more these elements are layered in, the harder it becomes to catch known or unknown vulnerabilities before deployment. By the time something breaks, it's often expensive to fix, or even worse, the embedded system is already in the field. This is where the concept of shifting security left comes in.

Rather than treating security as a final-stage check, shift left pushes visibility and validation earlier in the development process, starting at the code, component, and build levels. For your organization building firmware, this approach is critical to reducing downstream risk and ensuring product reliability.

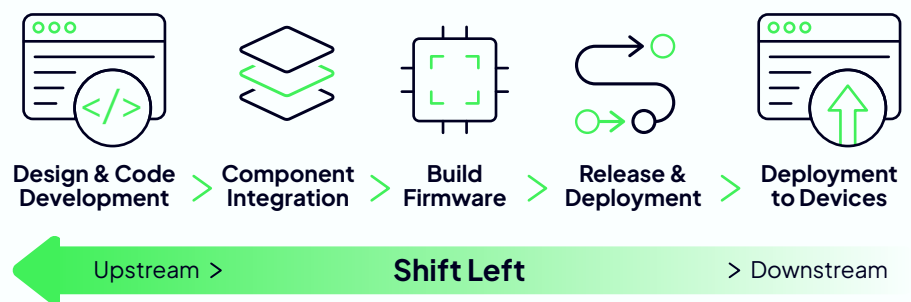


Figure 5: Shifting left focuses more on security measures near the upstream compared to the downstream

Concept of Shift Left

At its core, shift-left means moving security and quality controls upstream, closer to when code is written, components are integrated, and firmware is built, rather than leaving risk detection for the very end.

In embedded system supply chains, teams don't always have access to the source code. Firmware often arrives as compiled binaries from third-party vendors or suppliers, and the necessary code or scripts to build firmware are often AI-generated for faster development, however, with limited visibility into dependencies. Because of this, traditional security checks are frequently skipped or delayed. Consequently, vulnerabilities, licensing issues, or outdated components can flow unchecked into deployment.

Shifting left changes that. It focuses on early validation, repeatable policy enforcement, and proactive decision-making, allowing teams to stop risk before it becomes a liability.

Need in Software Supply Chain

As discussed earlier in the introduction section, the software supply chain isn't linear, but a complex web of internal teams, external vendors, legacy codes, etc. Due to the large number of stakeholders and moving parts involved, early-stage known and unknown vulnerabilities can remain unnoticed until much later. By this time they are more difficult and costly to resolve.

Shifting left addresses this problem directly by:

- Catching vulnerabilities and misconfigurations in early stages
- Preventing non-compliant code from entering the pipeline
- Enabling faster, more secure delivery with fewer late-stage surprises

The result is not only better security, but also shorter release cycles, fewer delays, and reduced costs associated with patching, rework, and product recalls. Ultimately, it increases customer trust.

Binary-Level Visibility Comes First

Reading this whitepaper up to this point, you've likely realized that reviewing source code is rarely feasible in embedded environments. Components may arrive as binaries without documentation, unclear licensing, and unknown contents. Due to this, binary-level visibility becomes the foundation of shift-left in embedded systems.

If you are wondering what a binary is, here is a simple explanation. A binary is compiled source code that has been translated into machine language, making it executable by a CPU. In embedded systems, firmware is typically made of these binaries along with scripts and configuration files that together control how a device operates.

When we talk about a firmware binary, it often means all of this is packaged into a single compressed image or archive. This package is what gets loaded onto the device.

Binary-level visibility allows teams to:

- Generate SBOMs even when the source isn't available
- Analyze third-party and vendor binaries before integration
- Detect vulnerabilities early and decide whether a component is fit for release

Without this level of insight, it's nearly impossible to enforce early-stage security, and shifting left becomes just a theory, not a practice.

Shifting security left is a mindset that requires the right tools and repeatable processes. In the next section, we'll explore the binary-level best practices that bring this strategy to life across your embedded supply chain.



SOFTWARE SUPPLY CHAIN SECURITY: BINARY-LEVEL BEST PRACTICES

Securing the software supply chain and putting shift left into action starts with adopting the right best practices at the binary level. These aren't just nice-to-haves; they are practical tools teams can rely on in day-to-day work. They help reduce risk, support compliance with new regulations, and build stronger firmware from the ground up. Along the way, they also bring clear benefits like faster delivery, fewer known and unknown vulnerabilities late in the process, and greater confidence in every release. The best practices are as follows:

Generate and Validate Software Bill of Materials (SBOMs)

SBOM is a list of everything that makes up your firmware. It includes libraries, components, versions, etc., either written by you, supplier-delivered, or AI-generated.

What it does: SBOMs help you understand what is inside a binary, and validation ensures the SBOM is accurate and complete.

Importance for Supply Chain Security and Shift Left: Libraries are often pre-compiled with no access to source code. SBOMs give visibility into these components so issues can be caught before being added to firmware.

Integrate Software Quality Gates

Quality gates are rules or security checkpoints that must be passed before you can move on in the development process.

What it does: These gates block binaries from being shipped into production that include critical vulnerabilities, unknown components, or missing SBOMs. You can customize gates based on your internal risk policies or regulatory requirements.

Importance for Supply Chain Security and Shift Left: When applied directly to release candidates of the firmware, quality gates help to identify and reject risky firmware at an early stage, before it reaches deployment to prevent business disruptions, establish a predictable release process, and protect brand trust.

Vulnerability Management and Impact Assessment

Identifying security issues in firmware by scanning the compiled binaries and understanding which vulnerabilities actually matter.

What it does: A good system not only flags vulnerabilities inside binaries but also identifies whether they are reachable, used, or exploitable in your environment.

Importance for Supply Chain Security and Shift Left: Binary-level analysis enables teams to assess risk early in the process, including for supplier-delivered firmware. Rather than being overwhelmed by lists of potential vulnerabilities, teams can focus on those that could threaten product integrity before release.

Maintain Lifecycle Visibility

Lifecycle visibility means tracking vulnerabilities in firmware and components, from build to deployment, even after it is in the field.

What it does: Even when working with compiled binaries, lifecycle visibility lets you track what's deployed where, what versions are running, and whether they include vulnerabilities or outdated components.

Importance for Supply Chain Security and Shift Left: Lifecycle visibility isn't just post-deployment monitoring. It also feeds insights back into earlier builds. If you know some components are troublesome or are risky, you can put them on a deny list and block them from being included in future releases.

Provide Flexibility – Custom Rules

Custom rules let your team define what “secure” looks like for your product, supply chain, and risk profile.

What it does: Instead of relying only on default scan rules, you can apply policies to binary-level metadata, such as banned libraries or supplier-delivered firmware components.

Importance for Supply Chain Security and Shift Left: Security isn't one-size-fits-all. Custom rules give you the power to catch problems unique to your supply chain or regulatory environment and catch them early.

Best practices only make an impact when they are consistently applied. This is only possible with automation.

AUTOMATING BEST PRACTICES FOR SOFTWARE SUPPLY CHAIN SECURITY

Best practices only work when applied consistently, where most teams struggle. Manual reviews are not scalable, particularly in complex environments where firmware comes from multiple vendors and AI tools arrive in binary form, and release cycles are short. Automation bridges this gap. It embeds security and compliance steps directly into daily development and integration workflows. Rather than relying on individuals to remember checks or chase documents, automation takes care of this.

Top Reasons to Select Automated Solution

- **Automated Binary-Level Analysis:** Most tools rely on either source code or a clean SBOM. However, these are not always available in embedded environments. Automated binary analysis overcomes this issue by directly analyzing compiled firmware to identify components, configurations, and libraries, without requiring transparency from upstream sources. This makes it possible to conduct security assessments even when the code is closed, supplied externally, or poorly documented.
- **Automated Validation of AI-Generated Code:** AI-assisted development tools are being adopted at a rapid pace, enabling developers to write and integrate code faster than ever, which comes at a cost: hallucinated packages, insecure defaults, and hidden vulnerabilities. Automated verification of AI-generated code and its dependencies ensures that what gets used is safe, trustworthy, and compliant. It simplifies the adoption of AI tools, allowing developers to move fast and securely, without introducing unnecessary supply chain risk.
- **Automating Risk-Aware Decisions Before Shipping:** Go/no-go decisions are often based on incomplete or unclear risk data. Automated systems can pair vulnerability detection with impact analysis to identify which issues are actually exploitable rather than just present. This approach helps prioritize the right fixes and prevent insecure builds early on without slowing releases due to irrelevant alerts.

- **Real-Time Visibility Into Deployed Firmware:** Security doesn't end with shipment. Automated monitoring tracks deployed firmware versions and identifies exposure to new known and unknown vulnerabilities over time. Rather than relying on manual re-checks or periodic audits, the system continuously scans the firmware, providing real-time insights when patching may be infrequent.
- **Policy Enforcement With Custom Rules:** Every organization has its own risk tolerance, compliance requirements, and technical environment. Manually applying these policies is time-consuming and inconsistent. With automation, however, security rules become part of the build and validation process, rather than being reviewed later. Whether it's banning outdated libraries, blocking unapproved licenses, or flagging specific supplier binaries, these rules are automatically enforced every time. This transforms your internal policies into reliable quality gates without slowing down your teams.
- **Built-In Compliance Assurance:** Manual audit preparation is time-consuming and often misses details. Automation captures everything, including SBOMs, vulnerability reports, rule violations, and firmware version histories, as part of the process. Teams are always audit-ready and can demonstrate full traceability for standards such as EU CRA, RED2, NIST, and others without having to document it retroactively.
- **Automated Detection of Zero-Days and License Risks:** Some risks are deeply embedded in the firmware, while others have not yet been disclosed. Automated binary analysis can help to uncover both. It identifies early signs of zero-day vulnerabilities by recognizing risky patterns or unsafe functions, even without a published CVE. At the same time, it scans all third-party and open-source components for their associated license, including those in supplier-delivered or closed-source firmware, to ensure nothing slips through unnoticed.

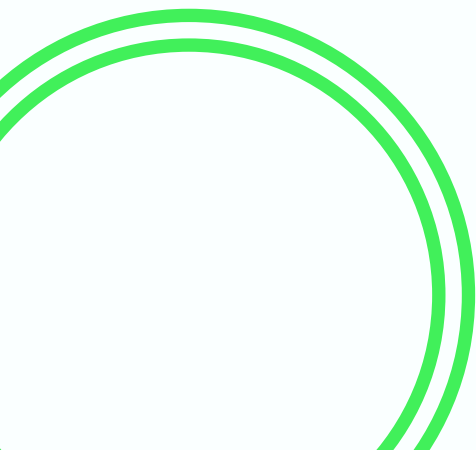


Benefits of Automation

To keep up the pace with modern-day firmware development, teams face growing complexity from third-party code, AI-generated suggestions, and opaque binaries. Manual processes simply can't keep up with the scale and speed of the modern software supply chain, especially in embedded systems. This is where automation becomes a game-changer. By embedding automated checks, policy enforcements, and others into every stage of firmware development and deployment, organizations can shift from reactive patching to proactive protection. The table below highlights what changes when you move to an automated solution.

With Automated Solution	Without Automated Solution
Automated SBOMs Generated from Binaries	SBOMs are missing or incomplete
Automated Verification of AI-Generated Code and Dependencies	AI-Generated Code Adopted Without Instant Validation
Automated Scans with Context-rich Triage	Manual Vulnerability Checks
Integrated Checks in CI/CD and Supplier Intake	Delayed Security Validations
Active and Automated Firmware Monitoring	Blind Spots Post-deployment
Proactive and Automated Auditable Vulnerability and Compliance Check	Reactive and Rushed Compliance

Securing the software supply chain isn't just a technical goal; it's a business advantage. Learn how our customer secured their supply chain.



ROI AND BUSINESS IMPACT THROUGH AUTOMATION

Securing the software supply chain doesn't have to slow things down; it can do the opposite. When done right and automated, it reduces operational risks, strengthens customer trust, and accelerates delivery cycles. ONEKEY brings measurable value by making firmware security more predictable, scalable, and aligned with real business goals.

How Swisscom Saves 400K USD per Avoided Incident

With over 1.8 million customer devices in the field, Swisscom faces high stakes every time a firmware upgrade is deployed. A single faulty rollout can cost the company an estimated CHF 374,000 (~400K USD) in support and repair. By integrating ONEKEY as a quality gate in its development and procurement workflows, Swisscom now performs deep analysis on over 80 firmware images annually, catching critical issues before they reach customers. This proactive approach not only avoids costly upgrade failures but also strengthens Swisscom's position in supplier negotiations and reinforces its brand as a security-first telecom leader. For Swisscom, ONEKEY isn't just a security tool; it's a measurable driver of operational efficiency, customer trust, and bottom-line ROI. Read the Full Swisscom case study here: <https://www.onekey.com/resource/swisscom>

ONEKEY's Platform Benefits

- **100% of the firmware supply chain is automatically inspected:** All internally developed or vendor-supplied firmware is automatically inspected for vulnerabilities, missing SBOMs, and license risks.
- **< 15 min for Firmware Analysis – Not days and Weeks:** Automated and binary-level scanning delivers complete firmware insights in minutes.
- **+60% reduction in false positives:** Context-aware scanning helps eliminate real risks while reducing noise and false positives that slow down teams.
- **10 or 10,000 firmware images — ONEKEY scales with you:** From niche devices to global fleets, ONEKEY supports secure intake and monitoring across product lines without extra overhead.

KEY TAKE AWAYS

- **Supply Chain Security is Essential:** Embedded products combine internal code, open source, supplier binaries, and AI-generated snippets, which increases blind spots and spreads attacks downstream.
- **Reality Check:** Partial analysis, missing SBOMs, and weak post-deployment monitoring let risks linger long after release.
- **Strict Regulation Enforcement:** CRA, RED, NIS2, EO 14028, FDA, IEC 62443, and others demand transparency, traceability, and accountability across the supply chain.
- **Shift Security Left:** Validating at the design, component, and build stages prevents problems from reaching staging, production, or the field.
- **Binary-Level Visibility is Necessary:** When source code isn't available, inspecting the firmware binary itself is the only way to know what's actually being shipped.
- **Automate the Critical Work:** SBOM creation, component vetting (including AI-suggested code), scanning, triage, and policy checks should run automatically in CI/CD and supplier intake.
- **AI Introduces New Traps:** Slopsquatting and "vibe coding" can pull in fake or unsafe packages, making automated dependency verification critical.
- **Security Beyond Shipment:** Continuous monitoring of deployed firmware helps map exposure to new CVEs and feeds lessons back into the next build.

Shift left, read the binary, and automate controls to turn supply chain security into a day-to-day practice.

Are you interested in further discussion with our security experts on how to achieve product cybersecurity compliance through effective and efficient SBOM management?

Please contact our security experts at:
experts@onekey.com - or call +49 211 1587 4104

READ FURTHER RELATED INSIGHTS

Understanding the EU Cyber Resilience act and achieve product cybersecurity compliance

This whitepaper explains the key requirements of EU Cyber Resilience Act (CRA) and how manufacturers, importers, and distributors of connected devices can meet its cybersecurity and compliance obligations. It covers product-level compliance, SBOM management and vulnerability handling that are crucial topic for supply chain. The readers will understand the roadmap for achieving CRA readiness and embedding continuous security validation across product lifecycles.

<https://www.onekey.com/resource/understanding-the-eu-cyber-resilience-act-and-achieve-product-cybersecurity-compliance>

SOFTWARE SUPPLY CHAIN REGULATIONS: How to achieve effective & efficient SBOM management?

This whitepaper provides a deep dive into how organizations can effectively create, validate, and maintain SBOMs to improve visibility and control in the software supply chain. It uses the regulatory landscape as context but centers on SBOMs as foundation for transparency, vulnerability detection, and supplier accountability. The readers will have a practical understanding oh how to operationalize SBOMs for better risk management and compliance efficiency across diverse software supply chains.

<https://www.onekey.com/resource/software-supply-chain-regulations-how-to-achieve-effective-efficient-sbom-management>

TACKLING SOFTWARE SUPPLY CHAIN RISKS WITH IEC 62443 AND SBOM

This whitepaper explores how the IEC 62443 framework can be applied to secure industrial software supply chains and how SBOMs support its implementation for component integrity and vulnerability management. It highlights risk management, and lifecycle security for vendors and operators across industrial automation, manufacturing, and critical infrastructure ecosystems. The readers will get a clear understanding of how IEC 62443 principles map to software chain security, helping organizations build compliant, resilient, and traceable OT and IoT environments.

<https://www.onekey.com/resource/tackling-software-supply-chain-risks-with-iec-62443-and-sbom>



onekey.com
+49 211 1587 41 04
info@onekey.com



© 2025 ONEKEY. All rights reserved. Reproduction only permitted with the approval of ONEKEY. All brands listed are the brands of the respective owners. Errors, changes, and availability of the listed products, services, characteristics, and possible applications reserved. Software & services will be provided by ONEKEY. ONEKEY makes no guarantee for the information of third parties regarding characteristics, services and availability. ONEKEY reserves the right to make changes to products and services as a result of product development, even without prior notification. None of the statements and depictions represents legal advice or may be interpreted in such a manner. In case of deviations from the contract documents and general terms and conditions of ONEKEY and their affiliated companies and subsidiaries in conjunction with this document, the contract documents and general terms and conditions always have precedent over this document.

V202510001EN