Software Supply Chain Management for Device Makers

Introduction

Modern embedded devices are no longer built on a simple, self-contained codebase like they once were. Instead, they're assembled from a complex web of software components: open source libraries, vendor board support packages and firmware, proprietary SDKs, and in-house code. Each component comes with its own **origin, license terms,** and **security posture,** including known vulnerabilities, project health, and maintenance status.

Together, these components form what is commonly referred to as the ***software supply chain.*** As devices continue to grow in complexity and scale, understanding and tracking this supply chain has become increasingly challenging. At the same time, the risks associated with poor visibility, from unpatched security vulnerabilities to license non-compliance, have grown significantly.

In this paper, we will explore the key processes required to track and manage the software supply chain effectively, and why doing so is becoming essential for device makers. We will look at how a well-managed supply chain can improve security posture, support regulatory readiness, and help ensure ongoing legal compliance throughout a product's lifecycle.

Key Takeaways:

Modern devices have complex software supply chains

Modern embedded devices combine hundreds of open-source, vendor, and proprietary components, making the software supply chain increasingly complex to manage.

Lack of visibility increases both security and compliance risks

Without a clear view of what software is inside a product, responding to vulnerabilities, meeting license obligations, or satisfying regulatory requirements becomes slow, reactive, and costly.

Open source enables innovation, but requires governance

Different license models and upstream maintenance practices demand structured processes to ensure compliance and reduce long-term IP and security exposure.

Software Bill of Materials (SBOMs) and standards are becoming a baseline expectation

SBOMs, particularly when aligned with standards such as SPDX and OpenChain, provide the foundation for vulnerability management, regulatory readiness, and transparent supplier relationships.

Continuous supply chain management supports secure, compliant products at scale

Integrating automated tooling, clear policies, and expert oversight into existing development workflows allows organizations to manage risk without slowing innovation.

What is really in your product?

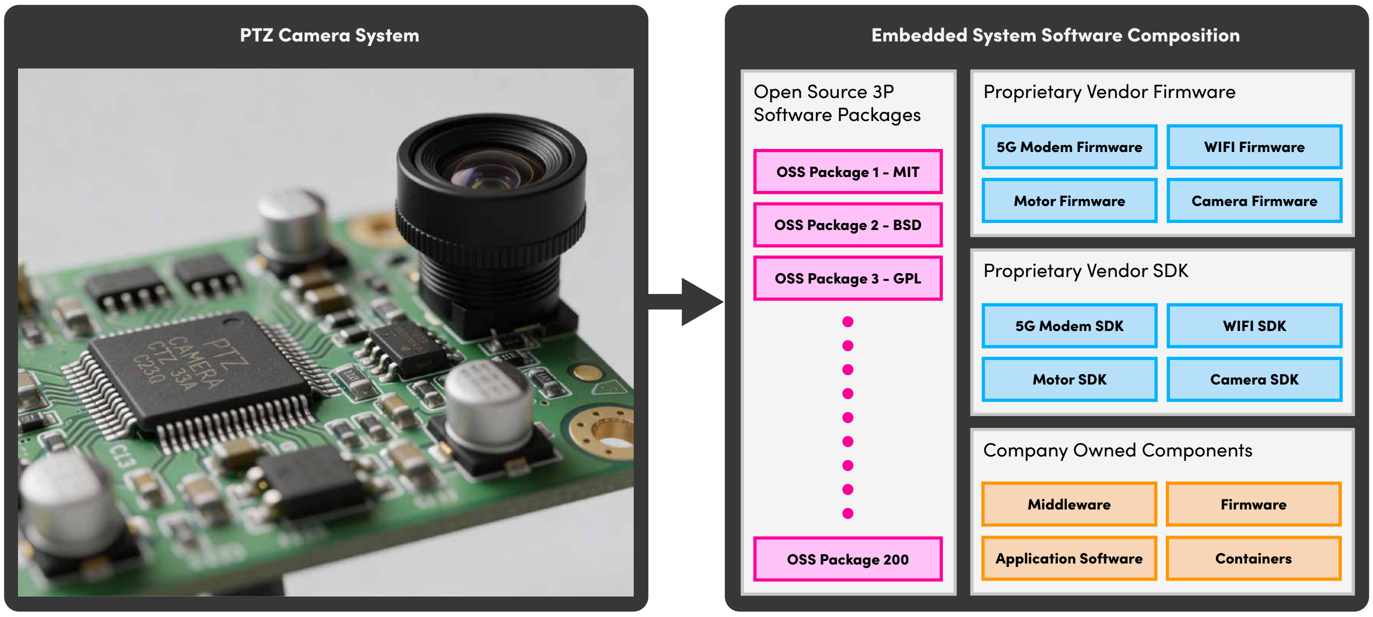
Even a relatively “simple” embedded device can incorporate hundreds of third-party software packages. Every hardware component or feature you add tends to introduce its own software into the system. For example, adding a Wi-Fi module means pulling drivers, firmware, and perhaps a vendor-provided software development kit (SDK). A camera module introduces image processing libraries. A cellular modem adds its own firmware and control interfaces. And so on.

Each of these components often relies on additional libraries (dependencies of dependencies), forming a deep and interconnected dependency tree. Crucially, **every component in that tree has the potential to introduce licensing obligations or security vulnerabilities,** whether it was written in-house or sourced from a third party.

To make this more concrete, consider a typical surveillance camera device. It might include a main processor running an operating system, along with a Wi-Fi module, a 5G modem, a pan-tilt-zoom (PTZ) motor, and a camera module. Each of those hardware elements comes with vendor-supplied software (drivers, firmware, SDKs) required to make it function. On top of that, the device maker will integrate additional open-source software, for instance, networking daemons to manage the Wi-Fi connection or media frameworks to handle camera streaming. Finally, the manufacturer adds its own proprietary middleware and applications to deliver the device’s unique features.

And remember: each of those components brings along its own baggage. There may be specific license requirements that need to be followed. There may also be known bugs or security flaws that could affect the product. All of these layers combined make up the software that ultimately ships inside the device.

The real challenge for device makers is getting a reliable handle on **everything that’s in the box** - because a hidden issue in just one upstream component can have serious consequences later.



Open Source Software Licenses: What They Mean in Practice

Open source software plays a fundamental role in most device software stacks. As we’ve already seen, a single product can include hundreds of third-party packages, many of which originate from open source projects. While this drastically accelerates development, it also introduces a critical consideration: **every open source package is distributed under a license, and each license comes with conditions that must be fulfilled:** if you do not fulfill the license conditions of a software component, you do not have the right to distribute that component, and the copyright holder may block the distribution of your product containing that component.

The Open Source Initiative (OSI) defines the criteria that open source licenses must meet, providing a common framework to reliably identify open source software that can be freely used, studied, modified, and redistributed under acceptable license conditions. In practical terms, when you ship a product, you are not just shipping code - **you are also bound to fulfill the legal conditions associated with every licensed component included in that product.**

Understanding Open Source Software License Types

Although there are many individual open-source licenses, most fall into a small number of broad categories. These categories are important because they determine how software can be redistributed and what obligations are triggered when a product is shipped.

License Types	Description
Permissive	Code may be distributed under a different license (including proprietary), with no requirement to release the source code
Copyleft (Weak)	If distributed, the relevant software component must be released under the same license, and source code made available; but combination with software under a different license (eg. proprietary) may be allowed, subject to certain conditions.
Copyleft (Strong)	If distributed, the relevant software component must be released under the same license, and source code made available; the copyleft extends to other components combined with or linked to it, so combination with proprietary software is not allowed.
Copyleft (Strong/ Weak SaaS)	Similar to Copyleft, except making the software's functionality available across a network counts as redistribution.
Proprietary	Any license that does not meet the OSI criteria cannot be called "Open Source" by definition. However, the component may be "open" by other criteria. Non-Free / Proprietary components are handled on a case by case basis as no other assumptions can be made about the rights granted to the licensee.

Common Open Source Software licenses

In practice, most embedded products rely on a relatively small subset of widely used licenses. These licenses appear repeatedly across operating systems, middleware, and common libraries.

Name	SPDX Indentifier	Type
3-clause BSD License	BSD-3-Clause	Permissive
Apache License 2.0	Apache-2.0	Permissive
GNU Affero General Public License version 3	AGPL-3.0-only	Copyleft (Strong SaaS)
GNU General Public License version 2	GPL-2.0-only	Copyleft (Strong)
GNU General Public License version 3	GPL-3.0-only	Copyleft (Strong)
GNU Lesser General Public License version 2.1	LGPL-2.1-only	Copyleft (Weak)
GNU Lesser General Public License version 3	LGPL-3.0-only	Copyleft (Weak)
MIT License	MIT	Permissive
Mozilla Public License 2.0	MPL-2.0	Copyleft (Weak)

Why license compliance matters

When a product is shipped, the distributor implicitly agrees to fulfill the license conditions of all included third-party software. Non-compliance is therefore not a theoretical risk; it can have very real technical and commercial consequences. **The main consequence is that the copyright holder of the affected software component could obtain an injunction to prevent you from continuing to distribute your product unless you completely remove that component from your product (which may prove impracticable in some cases) and also seek damages.**

In practice, potential impacts include:

- Legal disputes resulting in high financial penalties
- Loss of distribution rights until issues are resolved (if they can be resolved)
- Engineering rework to remove or replace problematic dependencies
- Reputational damage and loss of customer trust

Ultimately, license compliance is not a one-time activity. As products evolve, new software is introduced, dependencies change, and upstream projects are updated. Without active management, the risk of non-compliance grows over time. **Establishing a structured compliance process is key to avoiding these issues and ensuring that open source can be used safely and effectively at scale.**

Open Source Vulnerabilities: Staying Ahead of Security Risks

Licensing is only one aspect of the risk introduced by a modern software supply chain. **Security vulnerabilities in third-party software pose an equally significant, and often more time-critical, challenge.**

Modern devices depend heavily on open source software, which means they also inherit the security history of those projects. When vulnerabilities are discovered upstream, device makers need a reliable way to understand whether they are affected and how urgently they need to respond.

This is where the **Common Vulnerabilities and Exposures (CVE)** system comes in.

What is a CVE?

The CVE system provides a standard way to identify and track publicly disclosed security vulnerabilities in software components. Each CVE entry describes a specific weakness that can be exploited to negatively impact one or more aspects of system security: confidentiality, integrity, or availability.

Every disclosed vulnerability is assigned a unique CVE identifier. This allows vendors, developers, and security teams to discuss the same issue unambiguously, coordinate remediation efforts, and reference upstream advisories and patches with confidence.

Thousands of new CVEs are published every year. For device makers relying on large numbers of third-party and open source components, this has an important implication: **vulnerability management is not a one-off task - it is an ongoing operational requirement throughout the product lifecycle.**

Understanding vulnerability severity

Not all vulnerabilities carry the same level of risk. Once a CVE is published, its severity is commonly assessed using the **Common Vulnerability Scoring System (CVSS).**

CVSS is a standard vulnerability-severity rating system from 0 (least severe) to 10 (most severe). It provides a consistent baseline for comparing different issues across software components.

That said, CVSS scores alone don't tell the whole story. The real-world impact of a vulnerability often depends on how and whether the affected component is actually used in a specific product.

Why does this matter for device makers?

In isolation, a single vulnerability might appear manageable. However, modern devices can contain hundreds of software packages, each with its own vulnerability history. New vulnerabilities can be discovered **at any point in a product's lifecycle,** including long after devices have been shipped.

Without clear visibility into the software supply chain, organisations often struggle to:

- Identify whether a product is affected by a newly disclosed CVE
- Determine which version of a component is in use
- Assess whether a vulnerability is exploitable in their specific configuration
- Prioritise remediation efforts effectively

This challenge becomes even more apparent when vulnerabilities propagate indirectly through dependencies. This naturally leads to a broader class of threats: **software supply chain attacks.**

Software Supply Chain Attacks

As software supply chains grow in size and complexity, they have become an increasingly attractive target for attackers. Rather than attacking end products directly, software supply chain attacks target upstream components, tools, or processes that feed into many downstream products at once.

In practice, this can take several forms:

- Exploiting known or unknown vulnerabilities in open source components
- Malicious code injection into legitimate upstream projects
- Tricking build systems into pulling attacker-controlled packages, e.g., namespace attacks
- Compromising CI/CD pipelines, build infrastructure, or code-signing systems
- Gaining access to developer accounts or release tooling

What these attack types have in common is their reliance on implicit trust - trust in upstream software, build systems, and update mechanisms that are assumed to be safe.

That trust is what allows modern software development to scale. It's also what makes supply chains such a powerful attack surface.

High-profile incidents like **Log4Shell** and **SolarWinds** emphasize how a single supply-chain weakness can impact thousands of products. Log4Shell revealed that a flaw in one widely used library can cascade through countless devices - especially when organisations lack visibility into where that code runs. Similarly, the SolarWinds breach showed that compromising a trusted software update can invisibly infect thousands of downstream systems, even in critical infrastructure.

Taken together, these incidents reinforce a critical point: **security is no longer just about the code you write, but about the software you consume and the processes you trust.**

Without clear insight into upstream components, dependencies, and update paths, organisations are left reacting to incidents rather than proactively managing risk. Understanding what software is in a product, where it came from, and how issues might propagate through the supply chain becomes a prerequisite for effective security, not an optional extra.

Tracking the Software Supply Chain

So, how can device makers keep track of all of this in practice?

For most device makers, the challenge is scale. A single product can include hundreds of software components, often spread across multiple subsystems. In some cases, individual subsystems, such as 5G modem modules, may themselves run an entire Linux distribution. Each of these components introduces its own licenses, dependencies, and vulnerability history.

Critically, this landscape is constantly changing. New vulnerabilities are disclosed throughout a product's lifecycle, long after initial release. New software versions are introduced, suppliers change, and regulatory expectations evolve. As a result, software supply chain management cannot be treated as a one-off activity tied to a single release. It must be **continuously managed** throughout the product lifecycle.

Why manual tracking breaks down

Manually tracking software components might appear feasible early in a project, but it quickly becomes unmanageable as products mature. Keeping spreadsheets or ad-hoc records in sync with the actual software being built and shipped is both time-consuming and error-prone.

Without automation and standardisation, organisations often struggle to answer basic but critical questions:

- Which exact components and versions are included in the product?
- What licenses apply, and what legal conditions do they introduce?
- Which known vulnerabilities affect this specific configuration?
- How do vulnerabilities or license issues propagate through dependencies?

The difficulty is compounded when different teams, suppliers, or tools describe the same components in conflicting ways. To manage the software supply chain effectively at scale, **everyone needs to speak the same language.**

Standards as the foundation for tracking

This is where industry standards play a central role. Instead of creating custom tracking methods, device makers are adopting established standards for structuring and sharing software supply chain data.

The OpenChain Project provides this foundation through internationally recognised ISO/IEC standards. OpenChain focuses on defining the processes and governance needed to consistently manage open source compliance and security assurance.

Two standards are particularly relevant:

- **ISO/IEC 5230** for open source license compliance
- **ISO/IEC 18974** for open source security assurance

Together, these standards define what it means to have a mature, repeatable supply chain management program - covering policies, roles and responsibilities, review processes, and continuous improvement.

Software Bill of Materials: Making the supply chain visible

A key enabler of supply chain tracking is the **Software Bill of Materials (SBOM).** At a high level, an SBOM is a structured inventory of all software components that make up a system or product.

For embedded devices, this means capturing:

- All third-party and open source packages
- Proprietary vendor firmware and SDKs
- Company-developed middleware and applications
- The relationships and dependencies between these components

An effective SBOM provides the foundation needed to manage both compliance and security. Without it, organisations are left reacting to issues without a clear understanding of their exposure.

SPDX: Standardising SBOM data

To ensure SBOMs can be generated, exchanged, and consumed consistently, the industry relies on **SPDX (System Package Data Exchange),** an international open standard (ISO/IEC 5962).

SPDX defines a standard data model for describing software components, including:

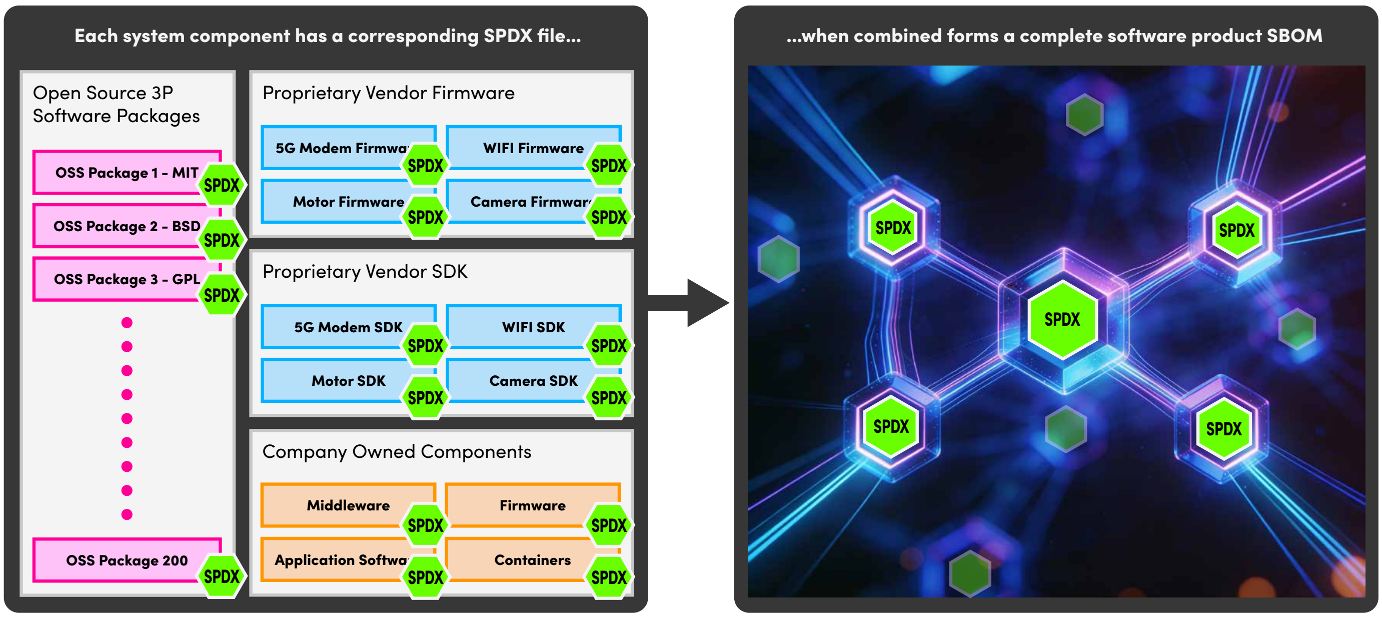
- Package metadata (name, version, description, etc.)
- License information
- Dependency relationships
- Associated vulnerabilities

While SPDX originated as a licensing-focused initiative, it has evolved to support a much broader range of use cases, including vulnerability management, regulatory compliance, and supply chain transparency. Its strength lies in enabling automation and interoperability across tools, build systems, and organisations.

Modern embedded build systems, such as Yocto-based environments, can generate SPDX data automatically as part of the build process. This makes a best effort to ensure that SBOM information reflects what is actually built and shipped. In practice, this relies on all software component metadata being completely accurate, which is often not the case. The complexity lies in managing components that may have incomplete or inaccurate information, for example, components that may fetch dependencies outside the visibility of the build system or components that have incorrect license definitions.

A system-wide view

In practice, supply chain tracking operates at multiple levels. Each software component can produce its own SPDX document that describes its metadata, licenses, and vulnerabilities. When combined, these individual documents form a **system-wide SBOM** representing the complete product.



Consuming SBOM data at scale

Generating SBOMs is a crucial step, but it is only part of the solution. A complete SBOM for a modern embedded product can be enormous, often consisting of tens of thousands of lines of data.

To make this information useful, organisations need tools that can consume SPDX data and present it in an accessible way. Typical requirements include:

- Filtering components by license type
- Identifying packages with known or unpatched vulnerabilities
- Linking vulnerabilities back to specific components and versions
- Supporting audits, reporting, and regulatory submissions

Standardised formats such as SPDX enable the implementation of tools and dashboards that can meet these needs, turning raw SBOM data into a more digestible format.

Software Supply Chain & The EU Cyber Resilience Act (CRA)

Tracking the software supply chain brings together everything discussed so far. It provides the visibility needed to manage license obligations, respond to vulnerabilities, and reduce exposure to supply chain attacks. More importantly, it establishes a foundation for continuous, proactive risk management across the whole product lifecycle.

The focus on software supply chain transparency is not happening in isolation. It is increasingly being driven by regulation. One of the most significant developments in this space is the **EU Cyber Resilience Act (CRA),** which introduces mandatory cybersecurity requirements for products containing digital elements that are placed on the EU market.

For device makers, the CRA represents a shift from best practice to **legal obligation.** Cybersecurity is no longer something that can be addressed solely through internal policies. It must be demonstrable, auditable, and maintained throughout the entire product lifecycle.

The CRA is designed to address the inconsistent level of cybersecurity in connected products and the lack of timely security updates once those products are deployed. To do this, the CRA introduces requirements that manufacturers must meet.

At a high level, the CRA requires manufacturers to:

- Design products with security in mind from the outset
- Understand and manage cybersecurity risks introduced by software components
- Remain continuously compliant by actively tracking, reporting, and fixing vulnerabilities post-market launch
- Provide transparent technical documentation to users and regulators

These requirements apply across the full lifecycle of a product, from development and release through to post-market maintenance.

Why supply chain tracking is foundational to CRA compliance

Meeting these requirements is tough without clear visibility into the software supply chain. Many of the CRA obligations depend directly on understanding what software is present in a product, where it originated, and how it evolves.

Software supply chain tracking enables CRA compliance in several ways:

Secure by design
Knowing which components are included, along with their security history, allows teams to make informed architectural decisions early in development.

Cybersecurity risk assessment
SBOMs expose third-party components and dependencies that may introduce additional risk, enabling more accurate and defensible risk assessments.

Vulnerability management
When a new CVE is disclosed, supply chain visibility enables quick determination of whether a product is affected and which versions are impacted.

Continuous compliance
Ongoing tracking ensures that post-release vulnerabilities are not missed, supporting timely remediation and update strategies.

Technical documentation
SBOMs and associated metadata provide a concrete, auditable foundation for the documentation required by the CRA.

In short, the CRA does not introduce entirely new technical challenges. Instead, it formalises expectations that already align closely with good software supply chain management practices. Organisations that have already invested in automated tracking, standardised SBOMs, and continuous monitoring are better positioned to meet these regulatory requirements.

Implementing an Effective Compliance Program with Linaro

Regulations like the EU CRA set clear expectations, but meeting them in practice can be challenging for device makers.

So far, we've already covered the foundations: understanding what software is in a product, tracking licenses and vulnerabilities, and using recognised standards like SBOMs and SPDX to bring structure to the supply chain. The remaining question is how to turn those foundations into something tangible that works day-to-day across real products.

At Linaro, we approach this through a phased compliance program that integrates policy, tooling, and continuous monitoring.

A phased approach to building a compliance program

Implementing an effective compliance program isn't the responsibility of a single team. It cuts across engineering, security, legal, and operations - and the order in which things are put in place matters.

A phased approach helps organisations establish the right foundations first, then build toward continuous compliance.

M1
M2
M3
M4
M5

Policy & Training Definition

Roles & responsibilities

List of licenses

Code acceptance & Procedure

Incident reporting procedure

Training guidelines

Toolchain Infrastructure Setup

Dedicated FOSSology Server install

Dedicated dashbord setup

Product repo software composition analysis

Build parser tooling integration

Initial Deployment

Full build and parsing through toolchain to extract license, CVE and repo metadata

Upload to FOSSology and harvest data to push to dashboard for SBOM visualization

Audit & Legal Support

Full legal audit of data by expert Open Source Compliance Lawyers

Identify risks in software supply chain

Continuous Deployment/ Compliance

Intergration into product CI/CD

Continuous harvest to keep dashboard up to date, ensuring continuous compliance

Policy & Training Definition

The foundation of an effective compliance program is a clear, shared policy. Linaro works with customers to define this policy in line with OpenChain standards, helping establish roles and responsibilities across the organisation - for example, who is accountable for open source decisions and legal sign-off. The policy also defines which license types are acceptable and sets practical criteria for introducing third-party software into products. This includes guidance on assessing the health and risk of upstream projects, such as whether a package is actively maintained or carries unresolved critical vulnerabilities. Alongside this, we help define incident reporting procedures, covering how customers and their users can report licensing or security issues. Linaro provides training materials to ensure the policy is applied consistently in practice. These materials can be rolled out internally so that engineering and product teams share a common understanding of expectations from the outset.

Toolchain Infrastructure Setup & Initial Deployment

Once policies are in place, Linaro works with the customer's IT team to establish the tooling infrastructure. We deploy a FOSSology instance (customer-hosted or Linaro-managed) and integrate Linaro's compliance dashboard into the customer's environment. Next, we integrate our build parser into the customer's build system (e.g., a Yocto build) to automatically capture license, provenance, and vulnerability metadata during each build. In scenarios where metadata is not reliably provided by the build system, the tooling is equipped to track such components through other methods. With this infrastructure ready, we run an initial build through the pipeline. The captured supply-chain data is processed in FOSSology to produce an initial SBOM with detailed software composition information. Those results are simultaneously fed into the dashboard, giving the team an easy-to-digest, system-wide view of the product's software components and their status.

Audit & Legal Support

Automated analysis provides valuable insight, but it has limits. In this phase, Linaro's expert auditors and open-source compliance lawyers review the supply chain data in detail, working directly from the dashboard and the underlying FOSSology data. Each software component is inspected to validate license accuracy and usage context, including cases where packages are dual-licensed or where certain license terms apply only under specific build or usage conditions. The review also looks for unresolved critical vulnerabilities, structural issues in the software composition, and potential intellectual property concerns such as copyright, trademark, or patent risks. The outcome is a clear, actionable report that highlights findings, explains their impact, and provides guidance on remediation where required.

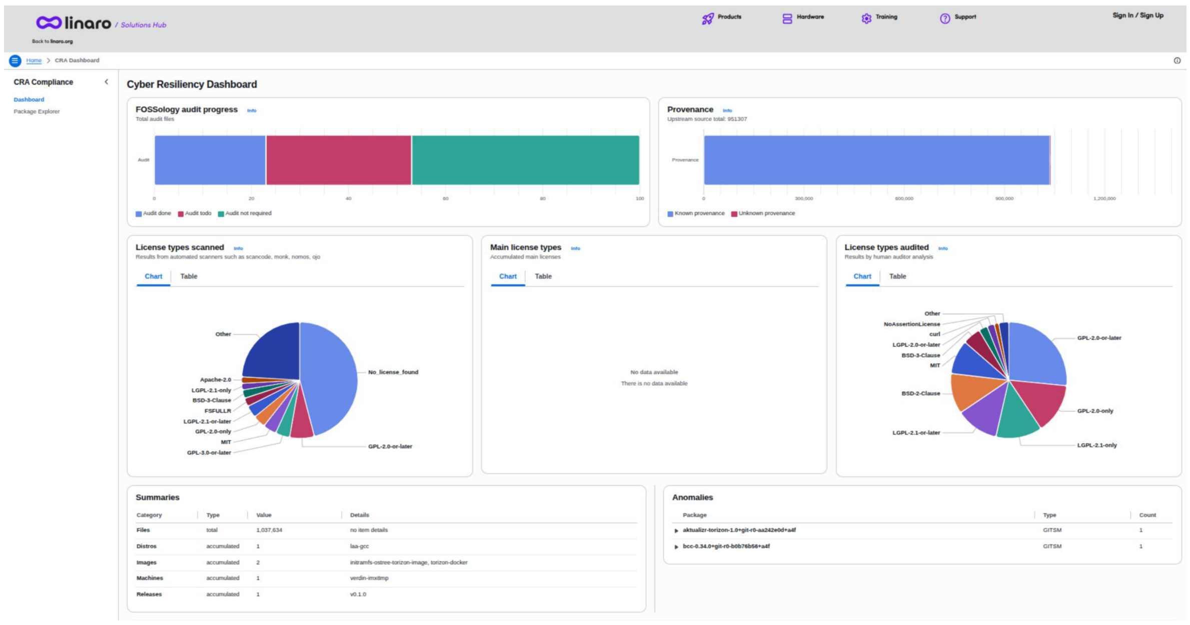
Continuous Deployment/Compliance

In the final phase, compliance becomes part of the standard release process. Linaro integrates the build-parser tooling into the customer's CI/CD pipeline, ensuring that each new build or release automatically passes through the supply chain analysis workflow. As new releases are produced, the dashboard is updated with the latest data, highlighting changes such as newly introduced packages, updated versions, or newly disclosed vulnerabilities. This delta-based view makes it easy for teams to focus on what has changed between releases, rather than re-analysing the entire system each time. Where required, these updates can also be fed back into the audit and legal review, ensuring compliance is maintained as software evolves.

Making the software supply chain visible

All of the phases described above are designed to achieve one key outcome: making supply chain data accessible, continuous, and usable in practice.

SBOMs can contain thousands of entries across hundreds of packages, and vulnerability data changes continuously, so working directly with the raw data is impractical for most teams.



Summaries

Category	Type	Value	Details
Files	total	1,037,634	no item details
Distros	accumulated	1	laa-gcc
Images	accumulated	2	initramfs-ostree-torizon-image, torizon-docker
Machines	accumulated	1	verdin-imx8mp
Releases	accumulated	1	v0.1.0

Anomalies

Package	Type	Count
▸ aktualizr-torizon-1.0+git-r0-aa242e0d+a4f	GITSM	1
▸ bcc-0.34.0+git-r0-b0b76b56+a4f	GITSM	1

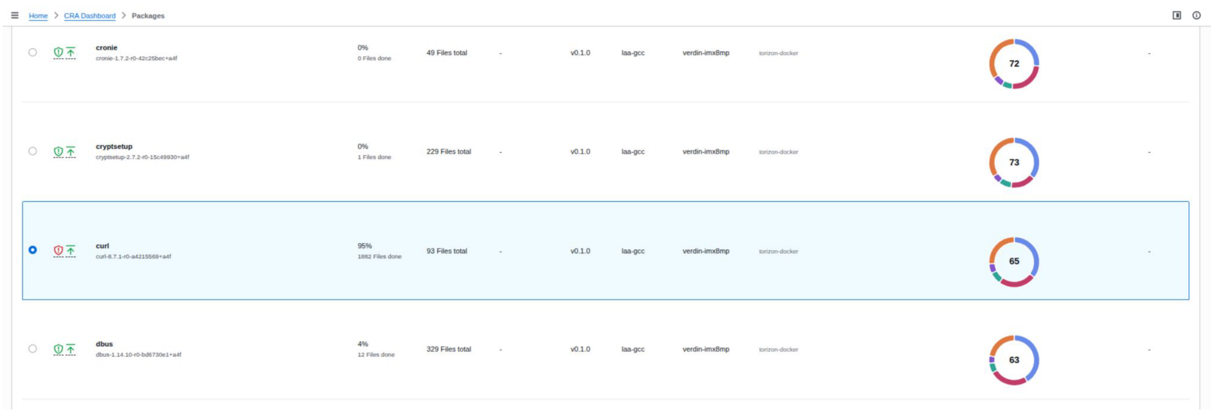
The dashboard provides a structured view of the software supply chain, allowing teams to quickly understand what is included in a product, where it came from, and what risks may be present. It enables stakeholders to explore the supply chain without interacting directly with raw SPDX files or analysis tooling.

In addition, the dashboard can aggregate and display metadata from multiple products. This interlinked approach has the benefit of revealing common, shared components between products/projects. This enables device makers to determine what products are impacted in the case of a new vulnerability surfacing and plan remediation actions accordingly.

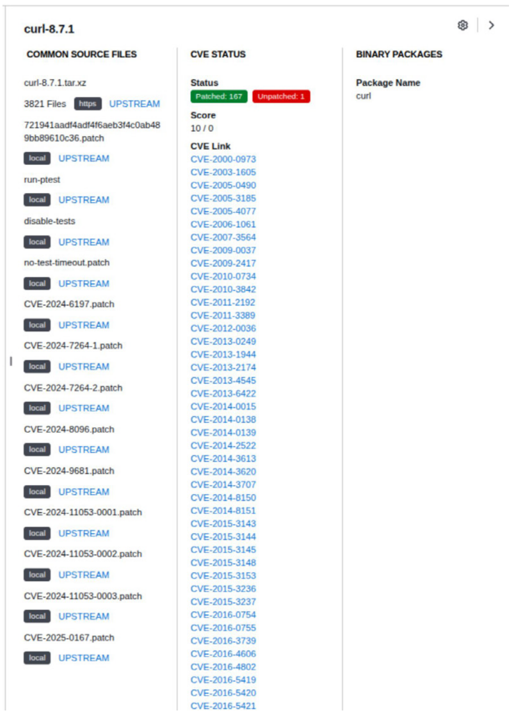
Common use cases include:

- Reviewing the complete list of software components and versions in a release and identifying possible software composition anomalies
- Understanding license distribution and identifying potential compliance risks
- Tracking known vulnerabilities and their status across the software stack
- Linking findings back to upstream sources, patches, and build metadata

By providing a shared, up-to-date view of the software supply chain, the dashboard supports collaboration across engineering, security, and legal teams. It also offers clear, auditable evidence for internal reviews, customer assurance, and regulatory compliance.



cronie cronie-1.7.2-r0-42c25bec+a4f	0% 0 Files done	49 Files total	-	v0.1.0	laa-gcc	verdin-imx8mp	torizon-docker	72	-
cryptsetup cryptsetup-2.7.2-r0-15c49930+a4f	0% 1 Files done	229 Files total	-	v0.1.0	laa-gcc	verdin-imx8mp	torizon-docker	73	-
curl curl-8.7.1-r0-a4215569+a4f	95% 1882 Files done	93 Files total	-	v0.1.0	laa-gcc	verdin-imx8mp	torizon-docker	65	-
dbus dbus-1.14.10-r0-bd6730e1+a4f	4% 12 Files done	329 Files total	-	v0.1.0	laa-gcc	verdin-imx8mp	torizon-docker	63	-


curl-8.7.1

COMMON SOURCE FILES

curl-8.7.1.tar.xz
3821 Files https UPSTREAM
721941aadf4adf4f6aeb3f4c0ab48 9bb89610c36.patch
local UPSTREAM
run-ptest
local UPSTREAM
disable-tests
local UPSTREAM
no-test-timeout.patch
local UPSTREAM
CVE-2024-6197.patch
local UPSTREAM
CVE-2024-7264-1.patch
local UPSTREAM
CVE-2024-7264-2.patch
local UPSTREAM
CVE-2024-8096.patch
local UPSTREAM
CVE-2024-9681.patch
local UPSTREAM
CVE-2024-11053-0001.patch
local UPSTREAM
CVE-2024-11053-0002.patch
local UPSTREAM
CVE-2024-11053-0003.patch
local UPSTREAM
CVE-2025-0167.patch
local UPSTREAM

CVE STATUS

Status
Patched: 167 Unpatched: 1
Score
10 / 0
CVE Link
CVE-2000-0973
CVE-2003-1605
CVE-2005-0490
CVE-2005-3185
CVE-2005-4077
CVE-2006-1061
CVE-2007-3564
CVE-2009-0037
CVE-2009-2417
CVE-2010-0734
CVE-2010-3842
CVE-2011-2192
CVE-2011-3389
CVE-2012-0036
CVE-2013-0249
CVE-2013-1944
CVE-2013-2174
CVE-2013-4545
CVE-2013-6422
CVE-2014-0015
CVE-2014-0138
CVE-2014-0139
CVE-2014-2522
CVE-2014-3613
CVE-2014-3620
CVE-2014-3707
CVE-2014-8150
CVE-2014-8151
CVE-2015-3143
CVE-2015-3144
CVE-2015-3145
CVE-2015-3148
CVE-2015-3153
CVE-2015-3236
CVE-2015-3237
CVE-2016-0754
CVE-2016-0755
CVE-2016-3739
CVE-2016-4606
CVE-2016-4802
CVE-2016-5419
CVE-2016-5420
CVE-2016-5421

BINARY PACKAGES

Package Name
curl

Conclusion

Modern device software is built on increasingly complex supply chains, and managing that complexity is no longer optional. What matters most is not just having the right tools, but putting the right processes in place to maintain visibility, reduce risk, and stay compliant as software evolves.

If you're looking to strengthen your approach to software supply chain management or want to sanity-check where you are today, Linaro is always happy to talk through the problem space and share what we've learned from working with device makers across the industry.

For more information speak with Linaro Experts

About Linaro

In 2010, Linaro was formed to unify a fragmented Arm software ecosystem. Our mission was clear: consolidate the code base to drive innovation on Arm. Today, that vision is a global reality. From Linus Torvalds releasing the kernel on Arm64 hardware to every cloud vendor having an Arm offering, the foundational challenges we set out to tackle have been solved.

As the industry matured, so did we. Linaro has evolved into a Services provider, leveraging our unrivaled Arm expertise to help customers build high-performing, compliant, and sustainable products.

While the way we collaborate has changed, our commitment to the open-source community remains the same. We continue to invest upstream and maintain critical projects, ensuring that every solution we build for our clients also strengthens the broader ecosystem.