Consultation Response

ENISA Security by Design and Default Playbook (v0.4, March 2026)

Submitted by: Bob Lord and Lauren Zabierek | Date: May 2026

Opening Remarks

The European Union and ENISA deserve genuine credit for making secure-by-design software a regulatory and policy priority. The Cyber Resilience Act represents a meaningful shift in how policymakers treat software safety, moving it from a matter of vendor goodwill toward a framework of manufacturer accountability. Documents like this playbook contribute to a global conversation that is long overdue, and the quality of thought behind the 22 principles, the CRA mapping in Annex C, and the machine-readable attestation concept in Section 5 reflects serious engagement with the problem.

The following comments are offered in that collaborative spirit. They are not intended as a critique of the document's technical content, which is largely sound, but as observations about strategic gaps that, if addressed, could make the final version significantly more impactful.

1. The Central Strategic Question: Why Does the Problem Persist?

This playbook joins a substantial body of existing guidance on how to build software more securely. OWASP, NIST, MITRE, and many national cybersecurity agencies have produced detailed, technically grounded resources covering much of the same ground. The document itself draws on many of them.

This raises a question the playbook does not address: if the information needed to build safer software has been widely available for decades, why do the same classes of coding errors persist?

Consider a concrete data point. MITRE's 2007 report, "Unforgivable Vulnerabilities," identified a set of coding errors so well-understood and so preventable that their presence in shipped software was described as indefensible. Nearly two decades later, the MITRE CWE Top 25 Most Dangerous Software Weaknesses list for 2025 contains many of the same classes of error, including cross-site scripting and SQL injection, that have had effective, widely documented mitigations available for the entire intervening period.

The persistence of these errors suggests that the limiting factor is not knowledge availability. Something else is preventing software manufacturers from acting on what is already known.

One well-documented contributing factor is the continued use of programming languages and development practices that make entire classes of vulnerability structurally possible. Memory safety vulnerabilities — the class that includes buffer overflows, use-after-free errors, and out-of-bounds reads — account for a significant share of critical CVEs across major software platforms. These are not errors that better training or more rigorous code review reliably prevents; they are errors that memory-safe languages make impossible by construction. The strongest version of secure-by-design software is not a product that scans its memory-unsafe C codebase thoroughly before shipping. It is a product whose architectural choices — including language selection, runtime environment, and component boundaries — eliminate entire categories of vulnerability at the design stage.

The playbook does not engage with this dimension of secure-by-design practice. Its Secure Coding Practices playbook (Section 4.9) treats security as a process discipline — follow the rules, run SAST, get a peer review — rather than a design discipline in which architectural choices determine what classes of defect are possible in the first place. That framing leaves a significant capability on the table.

A stronger version of the playbook would include explicit guidance on vulnerability class elimination: the use of memory-safe languages for new development, the migration of high-risk components in existing codebases, and the use of language and runtime features that eliminate injection vulnerabilities, type confusion errors, and integer overflow conditions at the compiler or interpreter level. This is consistent with guidance that the United States Cybersecurity and Infrastructure Security Agency, the National Security Agency, and the White House Office of the National Cyber Director have all issued in recent years, and ENISA is well-positioned to reinforce and extend it for European manufacturers.

More broadly, the persistence of known vulnerability classes despite decades of available guidance suggests that the limiting factor is not knowledge — it is incentive. Software manufacturers currently bear little direct financial or legal consequence for shipping products with preventable defects. The costs of insecure software are externalized onto customers, operators, and the public. Until that accountability structure changes, guidance documents, however well-constructed, are unlikely to change outcomes at scale.

Rather than adding another guide to the existing literature, ENISA is uniquely positioned to investigate the systemic factors and incentive structures that lead to pervasively unsafe software. Until those factors are understood, no playbook, however well-constructed, can reliably change outcomes at scale.

We recommend ENISA pursue two parallel efforts:

1. A broad survey of software manufacturers designed to surface how they currently measure the risks their customers face when deploying their products. The goal is to cast a wide net and generate data about current practices, priorities, and constraints.
2. A narrower program of structured interviews with a diverse set of software manufacturers to understand the systemic barriers to secure-by-design implementation. Unlike a survey, interviews allow for the kind of interactive, iterative conversation needed to identify root causes rather than symptoms.

The findings from these efforts should inform the next version of this playbook and, more importantly, ENISA's broader policy work on manufacturer accountability.

These two efforts should cast a wider net than software manufacturers alone. ENISA would benefit from examining the outcomes of government regulations that have historically targeted software operators rather than the makers of the software they deploy. Understanding why those regulations took the shape they did, and whether they produced the intended safety improvements, would sharpen ENISA's thinking about where accountability should sit in the software supply chain.

ENISA should also investigate the role insurance has played in improving safety in other sectors. In automotive, aviation, and construction, insurers have functioned as a powerful market-based force for risk reduction, pricing unsafe practices into premiums and creating direct financial incentives for manufacturers to improve. There is growing interest in whether cyber insurance could play an analogous role in software safety, and whether current insurance

market structures actually reward secure-by-design practices or simply redistribute the cost of insecure ones. Bringing these perspectives into the research, alongside the voices of software manufacturers, would give ENISA a more complete picture of the systemic forces shaping the current landscape.

2. The Document Would Benefit From an Explicit Outcomes Section

The playbook describes what software manufacturers should do. It does not describe what the world will look like if they do it.

A dedicated section defining intended outcomes would strengthen the document considerably. Specifically, ENISA should state: if software manufacturers adopt this playbook, what changes? Who benefits? By how much? Over what time horizon?

Being explicit about desired outcomes would serve several purposes. It would give the software sector a clear destination to navigate toward. It would give regulators and market surveillance authorities a basis for measuring progress. And it would give ENISA itself a mechanism for evaluating whether future versions of the playbook are working.

The absence of an outcomes section also makes it harder to prioritize among the 22 principles. Not all of them carry equal weight in terms of customer risk reduction. An outcomes-oriented framing would help manufacturers understand where to focus limited resources for maximum impact. A playbook organized around measurable outcomes — “implementation of this principle should reduce the frequency of credential-based compromises in the installed base” — would also create a basis for the kind of field feedback loops described in Section 6 of these comments.

Relatedly, the document would benefit from distinguishing between principles that reduce the probability of a vulnerability being introduced, principles that reduce the probability of a vulnerability being exploited, and principles that reduce the impact when exploitation occurs. These are meaningfully different objectives. A manufacturer working under real resource constraints needs to understand which principles deliver the greatest reduction in customer risk, and that understanding requires an outcomes frame that the current document does not provide.

3. Terminology Choices That Shape Accountability

3a. “Secure-by-Design” vs. “Security by Design”

The document alternates between “secure by design” and “security by design.” This is not a stylistic quibble. The choice of phrasing has real consequences for how accountability is assigned.

“Secure-by-design” functions as an adjective. It attaches to a noun: secure-by-design software, secure-by-design products, secure-by-design manufacturing practices. That attachment is important, because products have manufacturers, and manufacturers are the ones who should be held accountable for the safety of what they produce.

“Security by design” is an abstract noun phrase. It describes a concept without anchoring it to a product, a manufacturer, or a customer. Documents that use this phrasing make it easier to discuss security as an aspiration rather than a property of specific products for which specific parties are responsible.

This distinction matters beyond typography. The automotive safety analogy is instructive: the concept of a “crashworthy vehicle” anchors safety to a product and its manufacturer in a way that “vehicle safety by design” does not. The former creates legal and regulatory accountability. The latter describes a philosophy. ENISA’s document is trying to do the former, and its terminology should reflect that.

We recommend the document adopt “secure-by-design” as the standard form throughout.

3b. Whose Risk? Making “Risk Management” Explicit

The term “risk management” appears throughout the document, including in the phrase “risk-informed decisions,” without consistently specifying whose risk is being considered.

This ambiguity has practical consequences. When a software manufacturer reads “risk management,” they may reasonably interpret it as their own business risk, reputational risk, or regulatory compliance risk. The customer’s risk of deploying the software in a real environment may not be what comes to mind.

We recommend the document adopt more explicit language throughout. Phrases like “customer risk management” and “reducing the customer’s deployment risk” make clear that the manufacturer is performing the assessment, but with the intent of benefiting the people who depend on the software.

Making the customer explicit in every reference to risk management also surfaces a set of questions that the document does not currently ask:

- How does the manufacturer currently measure the risk a customer faces when deploying this product?
- What artifacts does the manufacturer collect from the field, via telemetry or structured studies, to understand how customers are actually using the software?
- How does the manufacturer track the most common attack patterns succeeding against their products in the wild, and what is their strategy for increasing the cost of those attacks to the point where they become economically unattractive to adversaries?

These are not rhetorical questions. They point to a significant gap in current secure-by-design practice: the absence of a systematic feedback loop from the field back into the software development lifecycle.

4. Accountability Requires Verification, Not Just Attestation

The playbook’s 22 principles are organized around checklists and release gates that manufacturers are expected to apply to their own products. The Machine-Readable Security Manifest concept in Section 5 extends this into structured self-attestation. Both are constructive contributions. But the document does not address the mechanism by which these self-assessments are verified, challenged, or enforced.

This is the gap between a compliance checklist and a safety standard. A manufacturer who ships a product with SAST scanning enabled, a vulnerability disclosure email address, and a completed release checklist has technically satisfied most of the playbook’s requirements. That manufacturer may still ship a product built on a memory-unsafe codebase with no plan to migrate, a vulnerability disclosure process with no binding remediation timeline, and a threat

model that was completed once at product launch and never revisited. The playbook's current structure cannot distinguish between these two manufacturers.

The CRA creates market surveillance obligations that could, in principle, close this gap. But the playbook does not engage with how its principles would interface with CRA enforcement in practice: what a notified body assessment of design processes looks like, how market surveillance authorities would evaluate a manufacturer's threat model, or what evidence would support a finding that a product was not, in fact, secure by design. Without that connection, the playbook risks functioning as a sophisticated checklist that manufacturers complete to demonstrate compliance rather than to improve outcomes.

We recommend the document include an explicit section on verification and accountability — describing how the principles in this playbook connect to CRA enforcement mechanisms, what independent verification of secure-by-design practices might look like, and what role procurement requirements and buyer-side due diligence can play in creating market demand for verified security claims. The Machine-Readable Security Manifest is most valuable not as a manufacturer-issued self-attestation, but as a procurement instrument that allows buyers, regulators, and integrators to independently evaluate security claims. That use case deserves explicit development.

5. From Machine-Readable to Machine-Usable

Section 5's treatment of machine-readable security attestation is one of the more forward-looking parts of the document, and the concept of a Machine-Readable Security Manifest is worth developing further. However, the framing around "machine-readable" understates the actual goal, and that gap has significant practical consequences.

The CVE program offers an instructive example. CVE records have been machine-readable for years; they are encoded in JSON and technically ingestible by any software. Yet a significant portion of CVE records contain fields that are populated with free-text descriptions rather than structured data. A field designed to capture affected version ranges may contain the phrase "all versions" instead of a specific range like 3.1.0 through 3.2.5. Organizations consuming CVE data have written thousands of lines of code simply to interpret the intent of analysts who used the wrong field or expressed structured information as unstructured prose.

Machine-readability is a prerequisite, not a destination. The goal should be machine-usability: data that can be ingested and acted upon by automated systems without human interpretation.

Achieving machine-usability requires a strongly typed schema in which every field has a defined type, a constrained set of valid values, and no escape hatch to free-form text for fields that should be structured. It requires validation at the point of creation, not just at ingestion. And it requires governance: a process for maintaining the schema over time as the threat landscape evolves and new security properties need to be expressed.

We recommend the document update its framing in Section 5 to reflect this distinction and to include minimum schema requirements that would allow automated processing without interpretation.

6. Active Voice and the Maker/Operator Distinction

The playbook occasionally blurs the distinction between software manufacturers and the operators who deploy their products. This creates ambiguity that can undermine accountability.

Consider this sentence from Section 4.13: “Vulnerability and patch management should be practical, repeatable, and prioritised by risk.” This sentence does not specify who is responsible. Is it the manufacturer creating a fix for a security defect in their product? Or the operator applying that fix in their environment? Both activities matter, but they involve different actors, different timelines, and different accountability structures.

Writing in the active voice with explicit subjects resolves most of these cases. “The manufacturer shall provide a security fix within X days of identifying a defect” and “the operator shall apply available security updates within Y days of release” are unambiguous in a way that passive constructions are not.

A systematic review of the playbook for passive constructions that obscure the responsible party would improve both clarity and accountability throughout.

7. Learning From the Field: Near Misses, Field Studies, and Continuous Improvement

The playbook’s treatment of operational practices would benefit from three additions that reflect how mature, safety-conscious industries approach continuous improvement.

7a. Field Studies

Software manufacturers routinely design products based on assumptions about how they will be deployed. Those assumptions are often wrong. Customers configure products in unexpected ways, operate them in environments the manufacturer did not anticipate, and encounter threat patterns that differ from what was modeled during development.

The playbook should include explicit guidance on how manufacturers can conduct structured field studies to learn how their customers actually deploy their products. This includes telemetry programs with appropriate privacy protections, structured customer interviews, and analysis of support cases as a signal of deployment problems. Without this feedback loop, secure-by-design principles are applied to an idealized deployment context that may not reflect reality.

7b. Near Misses

Near misses — cases where a critical vulnerability existed in a shipped product but was not exploited because of an accidental control, a slow-moving adversary, or simple luck — are among the most valuable data points available to a learning organization. They reveal real weaknesses in products that are already in the field, before those weaknesses result in harm.

Every software manufacturer should have a formal process for identifying, documenting, and learning from near misses. The playbook should call this out explicitly, including guidance on how near-miss findings should feed back into the software development lifecycle and risk management activities.

7c. Continuous Improvement and Executive Visibility

Secure-by-design is not a state that a product achieves and maintains automatically. It requires ongoing organizational commitment, including formal processes with executive visibility to drive improvement over time.

CISA's 2024 whitepaper, "Safe Software Deployment: How Software Manufacturers Can Ensure Reliability for Customers," offers a useful model for how this kind of organizational discipline can be structured. We recommend ENISA reference and build on that work.

The playbook should also address security incident simulations explicitly. Tabletop exercises that test how a manufacturer's leadership team would respond to scenarios such as a widely-reported critical vulnerability in an unsupported open-source library embedded in their product, or a supply chain compromise affecting their build pipeline, are among the most effective tools for identifying gaps in organizational readiness. It is far better to discover those gaps in a controlled exercise than during an actual incident.

Closing

The ENISA Security by Design and Default Playbook is a constructive contribution to a conversation the software industry needs to have more urgently. The suggestions above are intended to help ENISA move from articulating what good practice looks like to understanding why it is not yet universal, and to developing the tools and frameworks that can close that gap.

Two strategic additions would significantly increase the document's impact. First, incorporating explicit guidance on vulnerability class elimination — including memory safety, language selection, and architectural choices that make entire categories of defect impossible — would bring the playbook into alignment with the strongest current understanding of what secure-by-design software actually requires. Second, connecting the playbook's principles to verification mechanisms, procurement requirements, and buyer-side accountability would transform it from a manufacturer self-help guide into an instrument for market-level change.

The most important contribution ENISA could make at this moment is not another guide to secure development practices, but a rigorous investigation of the systemic factors that prevent those practices from being adopted. That investigation would position ENISA as a global leader in software safety, and its findings would give this playbook, and future iterations of it, the evidentiary foundation needed to drive real change.

References

- MITRE, "Unforgivable Vulnerabilities" (2007): https://cwe.mitre.org/documents/unforgivable_vulns/unforgivable.pdf
- MITRE CWE Top 25 Most Dangerous Software Weaknesses (2025): https://cwe.mitre.org/top25/index.html
- CISA, "Safe Software Deployment: How Software Manufacturers Can Ensure Reliability for Customers" (2024): https://www.cisa.gov/sites/default/files/2024-10/safe-software-deployment-how-software-manufacturers-can-ensure-reliability-for-customers-508c.pdf
- CISA/NSA, "The Case for Memory Safe Roadmaps" (2023): https://www.cisa.gov/resources-tools/resources/case-memory-safe-roadmaps
- White House ONCD, "Back to the Building Blocks: A Path Toward Secure and Measurable Software" (2024): https://www.whitehouse.gov/oncd/briefing-room/2024/02/26/press-release-technical-report/