The Trustworthiness Gap: Why Speed-to-Market Pressure Leaves Security Assurance Behind

The NIST SP 800-160 Perspective on Engineering Trustworthy Secure Systems

Dr. Ron Ross

RONROSSECURE, LLC

Introduction

Every engineering organization today lives under the same clock. Product roadmaps are measured in sprints, competitors ship weekly, and "time to market" has become as much a design constraint as cost or performance. Under that pressure, teams naturally gravitate toward the parts of engineering that produce visible, demonstrable progress including a defined set of requirements and a working solution that satisfies them. What gets squeezed out is less visible, harder to demonstrate, and slower to pay off — the discipline of proving, with evidence and reasoned argument, that the system can actually be trusted.

NIST Special Publication 800-160, Volume 1, Revision 1, *Engineering Trustworthy Secure Systems* [1], describes exactly this dynamic through its systems security engineering framework [2]. The framework defines three interacting contexts that every trustworthy secure[1] system must pass through: (1) the problem context, (2) the solution context, and (3) the trustworthiness context. In principle, the three are inseparable, looping back on one another throughout the system life cycle. In practice, under the weight of innovation pressure and speed-to-market incentives, organizations frequently emphasize the first two contexts while treating the third as an afterthought or skipping it almost entirely. That imbalance is not a minor process gap. It is the difference between a system that appears to be secure and one that has actually earned justified confidence.

The Systems Security Engineering Framework

NIST SP 800-160 frames the systems security engineering framework as a *conceptual view* of the key contexts within which systems security engineering activities are conducted. It is deliberately not a sequence of steps to check off, but a set of interacting contexts, each with its own checks and balances, meant to be executed in a closed loop iterative and recursive manner [1]. The three contexts are:

- **The problem context:** Understanding what *adequately secure* means for this system and these stakeholders.
- **The solution context:** Architecting, designing, and realizing a system that satisfies that understanding.

[1] The term *trustworthy secure* has a specific meaning in NIST SP 800-160. With the complexity of modern computing systems, a secure system is difficult if not impossible to achieve. The trustworthiness of a system can vary widely. It is based on the level of assurance that the system was designed to achieve and has demonstrated through *assurance evidence* and *assurance cases*.

- **The trustworthiness context:** Building and demonstrating the evidence-based case that the realized system actually meets the claims made about it.

The framework's own language is instructive: the first two contexts drive toward a solution; the third demonstrates the worthiness of that solution across competing and often conflicting constraints [1]. Skipping or shortchanging that third context doesn't just leave a document unwritten — it leaves the central question of the whole exercise unanswered: how do we actually know this system is trustworthy secure, and how confident should anyone be in that claim?

The Problem Context: Where the Clock Starts

The *problem* context defines the basis for an adequately secure system. It focuses on stakeholders' concerns about unacceptable losses given their mission, operational capability, and performance needs, along with the cost, schedule, performance, and risk-driven constraints that bound the effort. It includes defining security objectives, defining security requirements, determining measures of success, and determining life cycle security concepts.

This is comfortable territory for most organizations because it looks like product management. Objectives, requirements, and success metrics map directly onto backlogs, epics, and acceptance criteria which is the native language of agile delivery. A security objective can be written as a user story. A requirement can be tracked as a ticket. Because the problem context produces artifacts that plug directly into existing planning tools, it tends to survive schedule pressure reasonably well. It's the part of systems security engineering that already looks like "real work" to a delivery-focused organization.

The Solution Context: Where the Product Gets Built

The *solution* context establishes the security aspects and constraints for the architecture and design of the system that satisfies the problem context's requirements, realizes the design, and produces evidence that those requirements have been met. It rests on a balanced protection strategy that is preemptive and reactive, and it includes defining the security aspects of the solution, realizing those aspects during implementation, and producing evidence for them.

Like the problem context, the solution context also tends to survive innovation pressure because it is the *product*. Architecture decisions, security controls, encryption, access management, and system hardening are visible, can be demonstrated, and directly tied to shipping. Engineers get credit for building things. A security control that goes into the codebase is a line item stakeholders can see and a box a compliance checklist can tick. Even under aggressive deadlines, most organizations find a way to implement at least a baseline set of solution-context security features, because doing so is inseparable from building the system at all.

The Trustworthiness Context: The Part That Doesn't Demo Well

The *trustworthiness* context is different in kind, not just degree. NIST defines it as a decision-making context that provides an evidence-based demonstration through reasoning, that the system of interest is deemed trustworthy (or not) based on a set of claims derived from security objectives. It consists of developing and maintaining an assurance case and demonstrating that the assurance case is satisfied.

An assurance case is a well-defined and structured set of arguments and a body of evidence showing that a system satisfies specific claims [4]. Assurance cases provide reasoned, auditable artifacts that connect a top-level claim (or set of claims) to the evidence and explicit assumptions that support it [5]. Assurance cases are how organizations demonstrate complex emergent properties like security, safety, resilience, or reliability: not by asserting that a system is secure, but by building a compelling, evidence-backed argument for why it is *trustworthy secure*, tied back to stakeholder needs and expectations.

This is where speed-to-market culture runs into trouble. An assurance case does not ship a feature. It does not appear on a product roadmap. It requires "subject-matter expert analyses" to determine that every security claim is actually substantiated by evidence — work that is slow, specialized, and easy to defer. And critically, NIST is explicit that assurance cases must be maintained in response to variances throughout the engineering effort — meaning trustworthiness is not a one-time deliverable that can be produced at the end and filed away. It is a continuously maintained argument that has to keep pace with a system that keeps changing. That is precisely the kind of ongoing, unglamorous investment that gets cut first when a release deadline tightens.

> *Establishing the problem, solution, and trustworthiness contexts as key components of a systems security engineering framework helps ensure that the security of a system is based on achieving a sufficiently complete understanding of the problem as defined by a set of stakeholder security objectives, security concerns, protection needs, and security requirements. This understanding is essential to developing effective security solutions – that is, a system that is sufficiently trustworthy and adequately secure to protect stakeholder's assets in terms of loss and the associated consequences.* [NIST SP 800-160, Vol. 1]

Why the Imbalance Happens

Several forces compound to push organizations toward the problem and solution contexts while starving the trustworthiness context:

- **Visibility bias.** Sprint reviews and stakeholder demonstrations reward things you can show. A new feature, a new control, a passed penetration test — all visible. An assurance case, an argument structure, or a chain of substantiating evidence is not something you can put on a slide with a screenshot. Work that doesn't demo well tends to get deprioritized regardless of its actual value.
- **The "MVP" mentality applied where it shouldn't be.** Minimum viable product thinking is powerful for feature development but dangerous when applied to trustworthiness. There is no such thing as a minimum viable assurance case for a system handling sensitive data or safety-critical functions. The case either substantiates the claim being made, or it doesn't. Yet under schedule pressure, "we'll come back and document the assurance rationale later" becomes an easy, and usually permanent, deferral.
- **Assurance work resists parallelization.** Feature teams can be added to speed up delivery of the solution context. Assurance case development, by contrast, requires deep, continuous engagement with how the system actually behaves, what evidence actually exists, and whether the arguments connecting them actually hold up — work that doesn't compress neatly just by adding people.
- **Misplaced confidence from solution-context activity.** Because a functioning security control or a passed test event is tangible, it's tempting to treat those as sufficient proof of security. But NIST's framework is explicit that solution-context work and trustworthiness-context work are distinct. Implementing a control is not the same as demonstrating, with evidence and argument, that the

system actually behaves as intended and does not engage in anomalous behaviors under the range of conditions stakeholders care about. Organizations under speed pressure often conflate “we built it” with “we’ve proven it’s trustworthy” — precisely the gap the trustworthiness context exists to close.

The Cost of Shortchanging Trustworthiness

The framework’s closed-loop design means the three contexts are supposed to feed back into one another continuously — the output of one life cycle stage becomes the input to the problem context of the next. When the trustworthiness context is skipped or compressed, that feedback loop breaks. Variances in the system go unidentified because no one was maintaining the argument that would have surfaced them. Claims about security accumulate without evidence behind them, creating a kind of assurance debt that compounds the same way technical debt does — quietly, until it becomes very expensive to pay down, usually during an incident, an audit, or a certification deadline that can no longer be waved off.

The result is a system that looks secure on paper (i.e., objectives defined, requirements traced, controls implemented), but where no one can actually answer, with evidence, why a stakeholder should trust it. That gap tends to surface at the worst possible time including after a breach, during a post-incident review, or when a customer’s security team asks for the assurance case and finds there isn’t one.

Rebalancing the Framework

Treating the trustworthiness context as a first-class, continuously funded part of engineering, rather than an end-of-project audit, is the direction NIST’s framework points toward. In practice, that means building evidence-based, assurance case artifacts incrementally as objectives and requirements are defined, not after the system is built. It means tying release gates not just to “did we implement the requirements” but to “can we demonstrate, with evidence and argument, why these requirements satisfy the claims being made about them.” And it means giving trustworthiness work the same planning visibility as feature work, so that it competes for resources on its actual merits rather than losing by default to whatever produces a result during a sprint that can be demonstrated.

Speed to market and trustworthy engineering are not inherently at odds. However, they are at odds when trustworthiness is treated as the context you get to if there's time left over. NIST SP 800-160's own framing makes clear that the three contexts are meant to operate together, each checking and balancing the others. That balance matters most precisely where speed pressure is highest: mission-critical systems increasingly run on commercial components, software, and platforms that were never built with an assurance case of their own. An unproven trust claim in a consumer product is an inconvenience. The same unproven claim, inherited by a system controlling critical infrastructure, a weapons platform, or a medical device, is a liability that doesn't surface until the consequences are irreversible. An organization that consistently under-invests in the trustworthiness context isn't moving faster toward a secure system — it's moving faster toward a system whose security claims simply haven't been tested and quietly passing that risk on to whoever integrates its product into something that has to work.

References

[1] Ross, R., McEvilley, M., & Winstead, M. (2022). *Engineering Trustworthy Secure Systems* (NIST Special Publication 800-160, Vol. 1, Rev. 1). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-160v1r1

[2] McEvilley, M. (2015). Towards a Notional Framework for Systems Security Engineering. NDIA 18th Annual Systems Engineering Conference. (Cited in NIST SP 800-160 Vol. 1 Rev. 1 as the originating source for the systems security engineering framework's three-context structure.)

[3] National Aeronautics and Space Administration. (2011). System Safety Handbook, Volume 1: System Safety Framework and Concepts for Implementation (NASA/SP-2010-580, Ver. 1.0). https://ntrs.nasa.gov/api/citations/20120003291/downloads/20120003291.pdf

[4] Software Engineering Institute, Carnegie Mellon University.

[5] International Organization for Standardization/International Electrotechnical Commission. (2011). ISO/IEC 15026-2:2011 — Systems and Software Engineering — Systems and Software Assurance — Part 2: Assurance Case. https://www.iso.org/standard/80625.html

