Trustworthy System Control

Engineering Dependable Constraints on System Behavior Under Adversity

Dr. Ron Ross
CEO, RONROSSECURE, LLC
ISTS FELLOW, DARTMOUTH COLLEGE

Abstract

Trustworthy System Control is a foundational design principle for trustworthy secure systems. NIST SP 800-160, Volume 1, Revision 1 bases the design of system control functions on the properties of the generalized reference monitor. Protection mechanisms must be non-bypassable, evaluatable, always invoked, and tamper-proof. These criteria apply beyond conventional access control to mechanisms that constrain system behavior and outcomes. This article connects the principle to stakeholder needs, loss analysis, architecture, the system life cycle, engineering for compromise, complementary principles, and assurance evidence. The central proposition is that a required constraint is dependable only when the decision, enforcement, and supporting functions can be shown to preserve it under the stated conditions of adversity. Correct authorization alone does not establish safe or secure behavior, and a control that depends on a compromised system element's cooperation cannot independently constrain that element. The objective is to engineer and substantiate control functions that preserve required protection throughout operation, change, failure, and recovery.

1. The Engineering Problem

Modern systems depend on mechanisms that decide what may happen and enforce limits on what actually happens. An access service authorizes a maintenance operation. A controller accepts a new operating setting. An update mechanism admits replacement software. An AI agent invokes a tool that can change a mission system. Each decision can be correct according to its local rules while the resulting system behavior still produces unacceptable loss. The engineering problem is establishing which constraints must hold, where they are enforced, and why the mechanisms can be trusted to preserve them under adversity.

A protective setting is not the same as a dependable protective function. A maintenance policy may prohibit an unsafe change while a local interface accepts it. The system may check commands sent through one interface while allowing another interface to write directly to the actuator without those checks. A controller may enforce limits until a diagnostic mode disables them. A software update may replace the enforcement logic or alter the data on which its decisions depend. In each case, the apparent protection can be weaker than the architecture assumes.

> ***"A required constraint protects the mission only when the realized system can enforce it."***

The relevant systems include information technology, operational technology, Internet-of-Things devices, and cyber-physical systems. Their protection mechanisms can include hardware, firmware, software, physical devices, and human-supported processes. The principle applies to individual system elements and to mechanisms distributed across elements. A security gateway, physical interlock, separation mechanism, or protected recovery controller must be assessed according to the constraint it is intended to enforce and the conditions under which that constraint must remain effective.

System security measures often concentrate on identifying the requesting entity and deciding whether its request is authorized. Those decisions are necessary, but their value depends on effective enforcement. If a compromised process can bypass the mechanism, modify its policy, suppress its invocation, or corrupt its

© 2026 RONROSSECURE, LLC | CC BY 4.0.

supporting information, a favorable access-control test does not establish protection of the mission. The architecture must account for the mechanisms that make its decisions consequential.

The concern also extends to unintentional adversity. Incorrect inputs, resource exhaustion, sensor faults, timing errors, and incompatible updates can cause a controller to omit a needed action or issue a harmful one. A mechanism designed only to reject obviously unauthorized requests may not address these conditions. Protection needs should specify acceptable behavior when inputs are uncertain, supporting services fail, or several locally valid actions interact.

Frontier AI adds urgency to this problem. An agent may generate requests faster than a human can review them, combine several permitted operations, or manipulate information consumed by a more privileged service. A control implemented only as an instruction to the agent remains dependent on the agent's behavior. Consequential tool use requires appropriately protected enforcement at the interfaces and resources where effects occur. The relevant question is whether required constraints remain effective when the agent or another requesting system element becomes hostile.

The security design principle of *Trustworthy System Control* makes the basis for that confidence explicit. Engineers must connect the loss concern to a constraint, allocate its decision and enforcement functions, identify supporting dependencies, and substantiate the resulting mechanism. A product name, policy statement, or successful demonstration under cooperative conditions does not complete that argument.

2. What the Principle Means

NIST SP 800-160, Volume 1, Revision 1 states the security design principle of *Trustworthy System Control* as follows [1]:

> ***"The design for system control functions conforms to the properties of the generalized reference monitor."***

The reference monitor is an abstract concept for trustworthy mediation of access. It does not specify one product, operating system, or implementation. NIST distinguishes that concept from the reference validation mechanism, a realization involving hardware and software. *Trustworthy System Control* generalizes the concept to control and constraint-enforcing mechanisms that compose to provide system control functions. Its concern is the dependable realization of constraints, including those that support safety and security [1].

NIST identifies four essential design criteria in SP 800-160, Volume 1, Revision 1 and applies them to this principle. The following explanations retain their distinct purposes:

- **Non-bypassable.** The mechanism cannot be circumvented. A controlled operation must not reach the protected resource through an alternate interface, privileged path, or indirect mechanism that escapes the required restriction.
- **Evaluatable.** The mechanism is sufficiently small and simple to be assessed with adequate confidence. Analysis and testing of the mechanism must address the protection provided, the constraint enforced, and the correctness of the implementation. The required rigor should reflect the trust placed in the mechanism.
- **Always invoked.** The mechanism provides protection whenever its constraint applies. Suspension, omission, mode changes, and transitions must not create an interval in which a consequential operation occurs without the required protection.
- **Tamper-proof.** The protection functions and the data on which they depend cannot be modified without authorization. The scope includes enforcement logic, decision rules, configuration, and relevant

© 2026 RONROSSECURE, LLC | CC BY 4.0.

supporting state. Authorized modification still requires controlled change and renewed assurance that the required property continues to hold.

The non-bypassable and always invoked criteria are closely related but address distinct questions. The first asks whether an alternate path can evade the mechanism. The second asks whether the mechanism is active and applied whenever required along the relevant paths and states. One interface may check every command against the operating limits, while another can send commands to the controller without those checks. Conversely, even when all commands pass through a single interface, that interface may accept commands without checking them during startup or maintenance.

The term tamper-proof states a demanding design criterion. It should not be presented as an unlimited assertion that no conceivable adversary can modify anything. The assurance case must identify the mechanism's scope, authorized change paths, protection assumptions, and tolerated compromises. Protection of code without protection of the policy or configuration it uses does not satisfy the intended property.

NIST also distinguishes protection control decision functions, protection control enforcement functions, and protection control infrastructure functions [1]. A decision function determines the required restriction. An enforcement function makes that restriction effective. Infrastructure functions provide supporting services or information, such as protected storage or communication. Other system functions should not interfere with these protection functions. Their placement and privilege in the architecture influence the confidence that can be achieved.

"Define the constraint. Protect the decision. Enforce the result. Substantiate the mechanism."

Trustworthy System Control includes enforcing authorized and intended behaviors, protecting the control capability against targeted attacks, and avoiding self-induced emergent, erroneous, unsafe, or non-secure control actions. The principle therefore concerns more than stopping an intruder at an access point. It asks whether the control capability itself can be relied upon to preserve the specified behavior and outcomes under the conditions for which it is designed.

3. What the Principle Does Not Mean

Trustworthy System Control is easily confused with deploying security controls, implementing centralized management, or using a trusted product. Those choices can contribute to protection, but none establishes the principle by itself. The following distinctions help keep the engineering claim precise:

- **Trustworthy System Control does not mean selecting a control catalog establishes trustworthy behavior.** Catalog requirements can help specify protection, but the realized decision, enforcement, and infrastructure functions require architectural analysis and evidence.
- **Trustworthy System Control does not mean every protective function belongs in one central controller.** Centralization can concentrate authority and dependence. Distributed mechanisms can satisfy the principle when their composition preserves the required constraints.
- **Trustworthy System Control does not mean an authenticated request is necessarily acceptable.** A correctly identified and authorized entity can request an operation that would cause harm, supply unsafe parameters, or combine permitted actions in an unacceptable sequence.

© 2026 RONROSSECURE, LLC | CC BY 4.0.

- **Trustworthy System Control does not mean a monitor that reports violations enforces the required constraint.** Detection and response may support the design, but their timing and authority must prevent or limit loss within the specified bounds.
- **Trustworthy System Control does not mean a mechanism can establish its own independence by configuration alone.** If the system element being constrained can alter or disable the mechanism under the stated compromise model, the claimed restriction lacks an independent enforcement basis.
- **Trustworthy System Control does not mean tamper-proof mechanisms cannot be updated.** Authorized updates are legitimate, but they must preserve protection, constrain who may change what, and reestablish evidence that the modified mechanism continues to enforce the required property.
- **Trustworthy System Control does not mean denial is always protective.** Rejecting a required control action, disabling a safety function, or withholding essential service can itself cause loss. The specified protective response must account for mission continuity and physical safety.
- **Trustworthy System Control does not mean formal analysis or testing proves an unlimited claim.** Evidence applies to defined requirements, models, implementations, and assumptions. Unexamined interfaces, platform behavior, and operating modes remain part of the assurance problem.

The ability to evaluate the mechanism depends on understanding its scope and dependencies. A large collection of features does not become a trustworthy mechanism simply because it includes the desired setting. Complexity can obscure the paths and dependencies that determine effective control. Engineers should identify the smallest credible scope for the mechanism and include every function on which its enforcement claim actually depends—including platform services, management paths, and shared resources. Reducing the diagram's boundary without removing real dependencies does not simplify the assurance obligation.

An AI agent’s request to change a system setting illustrates the distinction between intended behavior and enforced behavior. A prompt can direct an agent to seek approval before a change. A separately protected tool interface can reject changes lacking the required approval and restrict the operation and parameters. These support different claims. Even the protected interface needs evidence that alternate tools, delegated identities, and management paths cannot evade its restrictions.

Finally, security is an emergent property of the system. Whether a control function delivers the intended protection depends on how it interacts with the rest of the system under the conditions in which it must operate. A correctly implemented protection mechanism cannot compensate for an incorrect assumption about sensor data. An intact software controller cannot enforce a physical constraint through an actuator that no longer responds as assumed. The engineering argument must explain the scope of the control function and the surrounding protections needed to address remaining loss pathways.

4. Loss-Driven Interpretation

Trustworthy System Control begins with the losses stakeholders consider unacceptable. NIST’s systems security engineering approach connects mission or business objectives and stakeholder protection needs to system requirements, architecture, design, and implementation [1]. Those connections establish what must be constrained. Starting with a preferred enforcement product can obscure whether the product protects the outcomes that matter or merely implements a familiar access rule.

Consider a water treatment facility in which a supervisory application requests changes to chemical dosing. Stakeholders identify two unacceptable outcomes within the example's scope: delivery of water outside defined treatment limits because of an unsafe dosing command, and interruption of minimum required treatment service because a supervisory fault or compromise disables necessary local control. The

© 2026 RONROSSECURE, LLC | CC BY 4.0.

supervisory application may legitimately request operating changes without being entrusted with unrestricted authority over treatment or independent protective functions.

A derived constraint might require every dosing change requested by the supervisory application to keep the process within an approved operating envelope. The constraint can specify permissible dosing settings, how quickly and in what sequence they may change, and the conditions that validated process data must confirm before a change is permitted. A separate continuity constraint can require necessary local control to continue or transfer to a specified protective mode when supervisory requests are rejected or the supervisory service is lost. The numerical limits and response times must come from facility-specific engineering and validation.

The control design should include how the operating envelope is established and changed. A compromised application that cannot change a dosing setting directly may attempt to widen the envelope, change the interpretation of process data, or disable command enforcement. The mechanism's policy and supporting information belong within the protection analysis. An unconstrained envelope administrator can invalidate a narrow claim about operational users.

The following activities connect the principle of *Trustworthy System Control* to loss concerns and provide a basis for allocating trustworthy control functions:

- **Identify the unacceptable outcomes.** State the operational and physical consequences that protection must address. Distinguish unauthorized changes from harmful but nominally authorized actions and from loss caused by omission or delay.
- **Derive observable constraints.** Specify the behavior, resource use, state transitions, timing, and operating conditions that must remain within bounds. Avoid requirements that call for a secure controller without defining the property to be enforced.
- **Trace the consequential paths.** Determine how requests, information, authority, and physical effects can reach the protected function. Include local interfaces, maintenance, updates, automated orchestration, and changes to governing policy.
- **Establish protective responses.** Define what happens when a request is invalid, information is uncertain, the mechanism cannot decide, or enforcement cannot be confirmed. Preserve the required balance of protection, minimum service, and safety.
- **State the adversity assumptions.** Identify the system elements that may fail or become hostile, the mechanisms that remain trusted, and the uncertainty the design must accommodate. Explain the consequences when those assumptions no longer hold.

An operating envelope is not automatically sufficient for physical protection. Several values may each lie within their individual limits while their combination is unsafe. A valid command may become inappropriate when delayed or applied in a different process state. Repeated allowed actions can accumulate into an unacceptable effect. Requirements and analyses should address the combinations and timing relevant to the identified losses.

The same reasoning applies outside operational technology. A financial workflow can require independent approval and transaction limits. A deployment service can require approved code and configuration before activation. An autonomous agent can be restricted to a set of task-specific operations. In every case, the constraint must describe the permitted effect and the conditions that justify it, not just the identity allowed to submit a request.

A useful engineering question is whether the constraint remains effective if the requesting system element becomes hostile at the moment it receives its legitimate authority. If the answer depends on that element

© 2026 RONROSSECURE, LLC | CC BY 4.0.

honoring instructions, the architecture needs a stronger allocation or a narrower claim. This connects *Trustworthy System Control* directly to *Engineering for Compromise* [2].

5. Engineering the Principle into the Architecture

The architecture must identify the complete control function before selecting implementation mechanisms. For each consequential constraint, describe its decision function, enforcement points, supporting information, infrastructure, and administration. Identify which resources are protected, which operations are controlled, and where evidence of effective enforcement can be observed. Trace each part to the protection requirement and the conditions under which it must hold.

The water treatment example uses a protection mechanism that checks supervisory dosing requests before they reach local dosing control. Its decision function evaluates the requested operation and parameters against the approved operating envelope and relevant process state. Its enforcement function prevents disallowed requests from reaching local dosing control. Local control and independent protective functions retain the authority needed to sustain the specified treatment behavior. This arrangement is illustrative. Its adequacy depends on the actual command paths, physical process, platform, and tolerated faults and compromises.

Enforcement should occur where the restriction can reliably govern the consequential effect. Checking a request at a user interface is insufficient if a backend service accepts equivalent commands through another channel. Validating configuration at submission is insufficient if it can be changed before activation. A protection decision and its enforcement must remain bound to the same operation, target, parameters, and relevant conditions.

The following architectural activities apply the four NIST criteria to the realization of system control functions:

- **Establish complete enforcement paths.** Identify every interface capable of the controlled effect, including direct device access, local consoles, maintenance connections, deployment services, and delegated operations. Remove unnecessary paths and apply equivalent protection to necessary alternatives.
- **Protect decisions and supporting data.** Control changes to policy, operating envelopes, identity evidence, configuration, process information, and timing sources. Specify how validity is determined and what happens when supporting information is unavailable or unreliable.
- **Separate protection from constrained functions.** Allocate enforcement and essential supporting functions so that a compromised requesting system element cannot modify or suspend them within its available authority. Analyze the platform and management dependencies on which that separation relies.
- **Minimize the mechanism's scope and complexity.** Include only the functions needed for its purpose where practical. Use clear interfaces and abstractions and keep unrelated functionality from inheriting the privilege or trust assigned to protection functions.
- **Preserve invocation through transitions.** Specify startup, shutdown, maintenance, emergency operation, update, and recovery behavior. Identify when the constraint applies and how equivalent protection is maintained before a new operating mode is initiated.
- **Constrain changes to the mechanism.** Define who can alter code, configuration, policy, and supporting data. Protect the update path and establish the analysis and verification required before a changed mechanism returns to service.

© 2026 RONROSSECURE, LLC | CC BY 4.0.

- **Provide a protective response to uncertainty.** Distinguish invalid requests from failed enforcement, missing information, and unavailable infrastructure. Specify bounded responses that preserve the required mission, security, and safety behavior rather than silently bypass protection.

A remote policy decision service can simplify administration but creates additional dependencies. The enforcement point must handle service outages, delayed responses, cached decisions, and changes in resource state. A previously valid authorization may no longer apply when the operation is executed. Engineers should specify currentness, revocation, disconnected operation, and the conditions requiring reevaluation. A policy decision delivered over a protected channel still needs correct interpretation and enforcement at the resource.

Tamper protection must include administrative authority. An application administrator may legitimately change application settings without being permitted to alter its enforcement mechanism. A platform administrator may nevertheless have authority over both. A signed update may be authentic but implement a weaker restriction. Where these authorities can invalidate a claim, either constrain them appropriately or make their trust explicit. Cryptographic verification of origin does not establish that a change preserves the required security property.

Hardware allocation can strengthen an enforcement boundary but does not settle every assurance question. Separate hardware may still depend on common management, power, restoration sources, communication, or clocks. A physical interlock can have a maintenance override. A device can enter an unprotected startup state before its firmware loads. Trace how every relevant dependency and mode affects the intended constraint rather than infer complete independence from physical separation.

Distributed control requires explicit composition. One mechanism may authenticate the request, another approve its content, and a third actuate the device. Their interfaces must preserve the meaning of the decision. Analyze replay, inconsistent policy versions, partial completion, retries, and changes between checking and use. A collection of locally correct mechanisms can still admit an unauthorized system-level effect when their assumptions do not align.

AI tool mediation presents the same architectural issues. A protected tool interface should govern the operation actually executed, including parameters, target resources, delegated authority, and task context. A second agent's approval is not independent if the requesting agent controls its inputs and credentials. High-consequence operations may require independently protected approval or validation. The model's response quality alone does not establish the trustworthiness of the surrounding control function.

> ***"The enforcement boundary includes everything that can change, bypass, or disable the restriction."***

6. Engineering for Compromise

Engineering for Compromise assumes selected system elements can act maliciously and examines which protections remain dependable [2]. *Trustworthy System Control* provides a design basis for the mechanisms expected to constrain that behavior. Its four criteria force attention to circumvention, interference, omitted protection, and adequacy of evidence. The compromise model determines which forms of self-protection and independence the architecture must support.

Start with the requesting system element. In the water treatment facility example, assume the supervisory application is hostile while local control, the protection mechanism, specified process data sources, and specified protective functions remain trusted. The adversary can use every interface and privilege available

© 2026 RONROSSECURE, LLC | CC BY 4.0.

to that application. It may submit unsafe parameters, repeat valid commands, replay approvals, exhaust resources, or exploit maintenance functions. Protection cannot depend on requests being well intentioned or submitted in the expected order.

Next, examine the control mechanism and its dependencies. An enforcement mechanism can constrain a hostile client while relying on a trustworthy platform, policy authority, or source of process data. Compromise of one of those dependencies may invalidate the enforcement claim. State that limit explicitly. If the architecture must tolerate compromise of a dependency, it needs an independent enforcement mechanism that remains effective under that compromise. Two mechanisms can both be defeated if the same compromised authority can disable or alter them.

The following scenarios help engineers distinguish the adversary's available authority from the control functions that must remain effective:

- **Compromised requester.** Exercise legitimate credentials and interfaces to attempt prohibited operations, harmful combinations, replay, delegation, and resource exhaustion. Observe the protected effect rather than only the request response.
- **Compromised supporting information.** Supply stale, contradictory, or malicious process values and authorization data. Determine whether validity checks and protective responses preserve the specified constraints under the faults included in the model.
- **Compromised decision authority.** Attempt to widen policy, alter operating limits, or grant authority that defeats protection. Identify which decisions require independent approval and which claims assume the authority remains trustworthy.
- **Compromised platform or administrator.** Determine whether the adversary can modify code, memory, configuration, invocation, or command routing. Identify the properties that remain independently enforced and those lost with the platform.
- **Compromised update or recovery source.** Attempt to restore obsolete policy, replace protection logic, or admit corrupted state. Verify that operational readiness requires reestablishing the approved control conditions.

A harmful request can be blocked while its indirect effects remain unbounded. Repeated rejected commands may fill logs, consume the enforcement mechanism's processing capacity, or starve local control. An application can submit a valid setting and then corrupt a data source used to interpret it. The analysis should cover both the specified decision and the effect of interacting with the mechanism. Minimum-service and timing requirements are part of protection where their failure contributes to loss.

Control functions can also create hazards through their own behavior. Protective mechanisms can issue conflicting commands. An abrupt denial can leave an operation partially complete. A fault response can disable necessary treatment rather than preserve it. NIST includes the avoidance of self-induced emergent, erroneous, unsafe, and non-secure control actions within the principle [1]. Security enforcement and safe system operation must therefore be analyzed together.

The objective is not to predict every exploit an adversary might discover. Engineers can establish constraints and analyze whether specified classes of behavior can violate them, including behavior that does not depend on a known attack signature. That supports reasoning across intentional and unintentional adversity. Confidence still depends on the completeness of the scope, the realism of the assumptions, and the evidence for the realized mechanism.

© 2026 RONROSSECURE, LLC | CC BY 4.0.

7. Interaction with Other Principles

Trustworthy System Control addresses the design assurance basis for mechanisms that enforce constraints. Its strongest relationships are with the security design principles that establish the required restrictions, protect their realization, reduce the assurance burden, and sustain protection under adversity. The following relationships explain their distinct architectural contributions [1]:

- **Mediated Access.** Specify and govern consequential interactions. *Trustworthy System Control* provides the design criteria for confidence that the mediation mechanism actually enforces its required decisions. A correct access policy needs a dependable realization.
- **Domain Separation.** Protect decision, enforcement, and infrastructure functions from inappropriate interference by other system elements. Identify management and platform paths that can defeat the claimed separation.
- **Least Privilege.** Limit the authority of requesting system elements and the protection mechanisms themselves. Restrict who can modify policy, replace code, disable invocation, or route commands around enforcement. Necessary high privilege increases the assurance obligation.
- **Least Functionality.** Reduce unnecessary capability in protection mechanisms. Unrelated diagnostics, scripting, administration, or data transformation functions can expand the paths that require analysis and the opportunities for interference.
- **Least Sharing.** Reduce common state and dependencies through which compromised system elements can influence control functions. Protecting the enforcement mechanism's code is insufficient if an untrusted system element can modify its policy store or exhaust its shared capacity.
- **Reduced Complexity and Clear Abstractions.** Support evaluation by making the mechanism and its interfaces understandable and assessable. The abstraction must describe the properties needed for enforcement without concealing relevant behavior.
- **Continuous Protection.** Sustain required protection whenever the constraint applies. Startup, maintenance, emergency modes, updates, and restoration must not create an unexamined gap in invocation or enforcement.
- **Distributed Privilege.** Allocate consequential authority so that a single compromised role cannot authorize and perform a change that defeats protection. Independence must extend to the mechanisms and credentials that enforce the distribution.
- **Protective Failure and Protective Recovery.** Preserve specified protection when control functions fail and when capability is restored. Define safe transitions and verify that recovered mechanisms enforce the approved constraints before operational use.
- **Compositional Trustworthiness and Substantiated Trustworthiness.** Establish the required properties of interacting mechanisms and support the claims with evidence. Component assurance and a list of deployed controls do not establish dependable system control by themselves.

These principles reinforce the four design criteria without replacing them. *Domain Separation* can protect the enforcement mechanism against a hostile application while a shared administrator retains the ability to disable it. *Least Privilege* can narrow normal operations while an update mechanism introduces broader functionality. *Continuous Protection* can require uninterrupted enforcement while a design leaves startup behavior unspecified. The combined argument must follow the actual relationships and system life cycle transitions.

NIST SP 800-160, Volume 2, Revision 1 provides related cyber resiliency techniques, including *Privilege Restriction*, *Segmentation*, *Substantiated Integrity*, and *Analytic Monitoring* [3]. These can support selected

© 2026 RONROSSECURE, LLC | CC BY 4.0.

architectural protections and operational evidence. Their use does not automatically establish the four criteria. Monitoring can reveal an enforcement failure without preventing the loss, and segmentation can leave the mechanism dependent on a common management authority.

The relevant distinction is between the property required and the means selected to achieve it. Engineers should state what each design principle or technique contributes, the assumptions on which it depends, and how its evidence supports the control claim. This keeps the architecture focused on mission protection rather than on the number of security features deployed.

8. System Life Cycle Application

Trustworthy System Control is a system life cycle obligation. NIST SP 800-160, Volume 1, Revision 1 provides processes for translating stakeholder concerns into requirements, architecture, design, implementation, operation, and disposal [1]. The control function's intended constraint and assurance basis should remain traceable throughout those processes. Late configuration of a protective setting may be unable to compensate for an architectural bypass or an unprotected management path.

The following activities show how engineers can establish and sustain trustworthy control functions throughout the system life cycle:

- **Business or Mission Analysis.** Identify mission behavior, unacceptable losses, minimum service, and the consequences of incorrect, omitted, or delayed control actions. Include intentional and unintentional adversity that can defeat protection.
- **Stakeholder Needs and Requirements Definition.** Establish protection expectations for operations, maintenance, emergency response, and recovery. Identify which activities may be constrained and which essential functions must remain available when a request is rejected.
- **System Requirements Definition.** Specify measurable behavioral constraints and the conditions under which they apply. Define required invocation, non-bypassability, authorized change, information validity, enforcement timing, and protective responses to uncertainty or failure.
- **Architecture Definition.** Allocate decision, enforcement, and infrastructure functions. Identify trust boundaries, supporting dependencies, administrative authority, alternate paths, and the protective functions that remain independent under the stated compromise model.
- **Design Definition.** Specify interfaces, state transitions, policy interpretation, resource allocation, update mechanisms, and self-protection. Specify how enforcement applies each decision to the same operation, target, and parameters. Reduce complexity and define abstractions that support rigorous assessment of the realized mechanism.
- **System Analysis.** Examine unsafe or non-secure control actions, timing, interference, common dependencies, and combinations of individually valid operations. Evaluate whether fault responses and degraded modes preserve the required balance of safety, protection, and minimum service.
- **Implementation.** Realize the specified constraints and protections. Prevent unapproved routes to consequential effects, protect control data and code, and remove unintended debug or override capabilities. Make the implementation traceable to the evaluated design.
- **Integration.** Verify that cooperating mechanisms preserve decision meaning and enforcement coverage. Examine identity propagation, parameter translation, platform behavior, retries, and policy version differences. Include supplier and enabling-system dependencies.
- **Verification.** Demonstrate the four essential design criteria and the specific constraint under the stated conditions. Combine analysis, interface testing, compromise-oriented exercises, fault injection, and implementation review as appropriate to the trust required.

© 2026 RONROSSECURE, LLC | CC BY 4.0.

- **Transition.** Establish approved code, policy, configuration, operating limits, and supporting data. Remove temporary installation and testing access. Verify that required protection is active before consequential operations are enabled.
- **Validation.** Determine whether the realized control functions meet stakeholder needs in representative mission scenarios. Include required maintenance, loss of supporting services, urgent operational changes, and safe recovery without bypassing protection.
- **Operation and Maintenance.** Monitor control availability, policy changes, exceptions, override use, and departures from the approved baseline. Reassess new interfaces, changed mission needs, supplier updates, and dependencies that can alter the assurance argument.
- **Disposal.** Withdraw control authority, credentials, and interfaces that are no longer needed. Address retained configuration, secrets, and restoration data. Ensure that retiring one system element does not disable continuing protection or leave a consequential control path accessible.

The life cycle processes of *Configuration Management* and *Information Management* maintain the approved mechanism, its operating assumptions, authorized changes, and supporting evidence. The *Risk Management* process informs the treatment of residual exposure and incomplete knowledge. The engineering record should show who accepted a limitation, which losses it affects, and what evidence is needed when the design or operating environment changes.

The *Acquisition* process should require evidence for the properties on which the system relies. Ask whether supplier mechanisms support complete mediation, protected policy and configuration, controlled update, invocation whenever protection is required, and assessment at the required rigor. Establish what the acquirer can independently verify. A supplier's assurance claim about normal operation may exclude the administrator compromise or maintenance mode that matters to the mission.

Changes require renewed analysis of the complete control function. Adding a tool, moving a service to shared infrastructure, changing a sensor source, or enabling remote support can introduce a bypass or a new trusted dependency. An update that leaves the visible policy unchanged can still alter enforcement timing or interpretation. Preserve evidence about the current realized system rather than relying indefinitely on the assurance of an earlier version.

9. Failure Patterns and Anti-Patterns

Failures of *Trustworthy System Control* often arise when an intended restriction is mistaken for an enforced property. The following patterns help focus reviews on the mechanism's effective behavior and the assumptions that can defeat it:

- **Policy without dependable enforcement.** A documented prohibition or user-interface check is treated as sufficient protection. Trace the controlled effect to every interface and establish where the restriction is made effective.
- **The constrained system element controls its own protection mechanism.** An application can disable enforcement, alter the limits used to evaluate requests, or modify the data on which the decision depends. Where the compromise assumptions require independent protection, place the mechanism and the data it relies on beyond the application’s authority.
- **An alternate path bypasses enforcement.** Local access, maintenance, deployment, direct storage writes, or a delegated service can produce the same effect without equivalent checks. Remove unnecessary paths and constrain the necessary ones.

© 2026 RONROSSECURE, LLC | CC BY 4.0.

- **Invocation stops during a transition.** Startup, diagnostics, emergency operation, or recovery suspends enforcement while consequential capability remains active. Specify equivalent protection and verify mode-entry and exit conditions.
- **Authenticity is mistaken for acceptable behavior.** A valid identity, signed request, or authentic update is treated as proof that its effect is safe and secure. Check the operation, content, context, and preservation of required constraints.
- **The check and the action refer to different conditions.** Parameters, targets, policy, or process state change after approval. Bind the decision to the executed operation and revalidate where relevant conditions can change.
- **Rejection becomes denial of essential service.** Invalid traffic consumes processing, fills storage, or triggers a shutdown that removes necessary local control. Verify resource bounds and protective continuity behavior under hostile use.
- **The controller creates the loss.** Conflicting commands, unsafe fallback, incorrect timing, or partial execution defeat the mission despite correct access decisions. Analyze the control action's physical and system-level consequences.
- **Complexity defeats assessment.** A general-purpose service carries unrelated privileged functions whose effects are not covered by the evidence. Reduce the mechanism's scope or expand the assurance boundary to include its real dependencies.
- **Weakened protection can go undetected after an authorized change.** An administrator or software update can alter policy, configuration, or logic in ways that weaken required protection. Reassess the affected assurance claims and verify that the modified mechanism still provides the required protection before returning it to service.
- **Recovery restores an unprotected state.** Recovered code loads obsolete rules, default credentials, or disabled validation. Restore the approved control conditions and check them before admitting consequential operations.

These patterns can combine. The protection mechanism may reject every invalid request through its public interface while an administrator changes its policy through a shared platform. The same platform can install an update that enables a local diagnostic path. A recovery process may then preserve that path because the approved baseline was never updated. Successful request tests alone will not reveal the complete loss of protection.

> ***"A control can make the right decision and still fail to control the system."***

Operational pressures deserve attention as well. If a protection mechanism routinely prevents necessary work, users may seek overrides or develop alternate routes. The appropriate response is to revisit the requirements and design, establish legitimate operational needs, and provide a controlled means of satisfying them. An exception process that can suspend essential protection without equivalent constraints undermines the architectural claim.

The review should distinguish a discovered vulnerability from a limitation already excluded by the claim. Both can matter to risk, but they require different treatment. A failed test within the tolerated compromise model challenges the claimed protection. An assumed-trusted administrator behaving maliciously identifies a limit of that model and may require a different architecture if the resulting loss is unacceptable.

© 2026 RONROSSECURE, LLC | CC BY 4.0.

10. Evidence and Assurance

A *Trustworthy System Control* claim requires evidence for the constraint and for the mechanism that enforces it. A policy setting, architecture diagram, or successful access test establishes only part of the argument. The assurance case must connect the specified behavior to decision, enforcement, self-protection, infrastructure, and the conditions under which the mechanism remains dependable.

A top-level claim might be:

> ***Within the defined system scope, the allocated control functions enforce the specified behavioral constraints under the stated operating conditions, system life cycle transitions, and failure and compromise assumptions. Their realization is non-bypassable, evaluatable, always invoked whenever the constraints apply, and protected against unauthorized modification of the functions and the data on which they depend.***

This claim is bounded by the constraints, system scope, and assumptions identified in the assurance case. It does not establish prevention of every unacceptable loss. The wider mission argument must also show that the constraints are adequate for the identified loss pathways and that physical behavior, supporting information, and other trusted functions satisfy their allocated obligations.

The following supporting subclaims organize the argument and identify evidence needed to substantiate the top-level claim:

- **Subclaim 1: Adequate and traceable constraints.** The constraints follow from stakeholder protection needs and loss analysis and cover relevant behavior, timing, and operating conditions. Evidence includes requirements traceability, control-action analysis, operational scenarios, hazard and security analyses, and stakeholder validation.
- **Subclaim 2: Complete and non-bypassable enforcement.** All paths capable of the controlled effect are subject to the required restriction. Evidence includes interface and authority inventories, architecture and data-flow models, direct and indirect path analysis, configuration inspection, code review, and adversarial attempts to reach the protected resource through alternate routes.
- **Subclaim 3: Always-invoked protection.** Protection remains active whenever each constraint applies, including defined mode changes and life cycle transitions. Evidence includes state-transition models, initialization and shutdown analysis, fault tests, maintenance and emergency-mode exercises, and observations of enforcement at the consequential resource.
- **Subclaim 4: Protected mechanism and supporting data.** Unauthorized changes to decision, policy, enforcement, configuration, and relevant infrastructure are prevented within the stated assumptions. Evidence includes privilege and interference analyses, platform protection evidence, protected update designs, administrative tests, integrity checks, and attempts to modify or suppress protection using compromised-element authority.
- **Subclaim 5: Evaluatable and correct realization.** The mechanism and its dependencies can be assessed at rigor commensurate with the trust placed in them. Evidence includes clear specifications and abstractions, design-to-implementation traceability, analysis of complexity and trusted functions, systematic testing, and formal analysis where warranted by consequence and feasibility.
- **Subclaim 6: Protective composition, failure, and recovery.** Interacting control functions preserve the required constraints without introducing unacceptable control actions. Evidence includes integration analysis, decision-binding tests (enforcement applies the decision to the same operation, target, and parameters), representative mission validation, capacity and timing results, fault injection, and recovery demonstrations that reestablish approved protection.

© 2026 RONROSSECURE, LLC | CC BY 4.0.

Non-bypassability requires analysis as well as tests. Enumerate the interfaces and authorities that can produce the protected effect. Examine maintenance, local access, administrative operations, storage access, deployment, and recovery. Tests should exercise these paths with the authority available to the compromised system element. Where a path is excluded from the model, identify the exclusion and its consequence for the claim rather than treating it as unimportant.

Showing that protection is always invoked requires evidence across operating states and transitions. Demonstrate that controlled capability is not available before protection is ready, that mode transitions do not suspend required enforcement, and that loss of a supporting service does not silently permit unrestricted operation. Observation of the protection mechanism's health indicator does not prove that each consequential operation passes through the required checks. Evaluate the interface and state where the effect actually occurs.

Tamper protection requires a complete inventory of what determines the restriction. Include executable code, policy, configuration, limits, authorization evidence, process data, and relevant timing or sequencing state. Attempt unauthorized modification, rollback, replacement, and disabling through the included compromise scenarios. Authorized changes need evidence of preserved properties and appropriate readiness checks. Protecting an old baseline faithfully is insufficient when that baseline no longer reflects the current requirements.

Explain how the mechanism and the functions it depends on can be tested and evaluated with confidence commensurate with the trust placed in them. Identify which functions are within the assurance boundary and explain how the evidence covers their contributions. A mechanism that depends on a large platform can require platform evidence even when its own code is small. Proofs and model-checking results establish properties within the scope of their models and assumptions. Tests provide evidence about the conditions actually exercised. Show how each form of evidence applies to the system’s implementation, configuration, and operating environment.

The water treatment facility example makes the argument concrete. Assume the design must tolerate a hostile supervisory application, invalid supervisory commands, and loss of the supervisory service. The protection mechanism, necessary local control, specified process data sources, and independent protective functions remain trusted. The example's two loss concerns are unsafe dosing caused by supervisory commands and loss of minimum treatment service caused by supervisory interference. These assumptions deliberately bound the claim; compromise of the trusted local controller requires separate analysis and protection.

For unsafe dosing, evidence must show that no supervisory path can issue or induce a command outside the validated envelope. Include direct commands, changed limits, out-of-date approvals, repeated actions, and indirect changes through configuration or management. Validate that the envelope and protective functions address the physical process conditions relevant to the claim. A range check alone cannot substantiate water safety when combinations, timing, or uncertain measurements can make an otherwise valid value unsafe.

For minimum service, evidence must show that rejected commands, traffic saturation, and loss of the supervisory service do not disable necessary local control. Test queue and processing limits, resource allocation, communication loss, mode transfer, and protective fallback under the specified conditions. Demonstrate that the protection mechanism's response does not remove the service it is meant to protect. Where continued normal treatment cannot be assured, specify the validated protective state and distinguish it from a claim of uninterrupted minimum service.

© 2026 RONROSSECURE, LLC | CC BY 4.0.

Recovery evidence must show that the protection mechanism returns with approved code, policy, configuration, and valid supporting state. Attempt recovery from interrupted updates, obsolete policy, and corrupted operational inputs within the defined test scope. Verify that specified paths that can change control settings remain constrained before reconnecting the supervisory application. A successful restart does not establish trustworthy recovery if required protection is temporarily disabled or the same compromise can rewrite the restored restriction.

> ***"Assurance must show both that the constraint is adequate and that its enforcement is dependable."***

Maintain the argument through change. New interfaces, altered authority, updated AI tools, supplier changes, and new operating modes can invalidate prior evidence. Operational observations help reveal drift and unexpected interference, but an absence of observed violations does not establish complete policy enforcement. The assurance case should identify known limitations, unresolved uncertainty, and the conditions requiring renewed analysis and verification.

11. Key Engineering Takeaways

The following takeaways summarize the engineering decisions and assurance obligations needed to apply *Trustworthy System Control* throughout the system life cycle:

1. *Trustworthy System Control* applies the generalized reference monitor properties to system control functions and constraint-enforcing mechanisms, beyond conventional access control.
2. Derive the required constraints from unacceptable losses, stakeholder protection needs, and intended mission behavior. Correct enforcement of an inadequate constraint cannot establish the required outcome.
3. Apply all four NIST criteria: non-bypassable, evaluatable, always invoked, and tamper-proof. Explain the distinct evidence supporting each criterion.
4. Include decision, enforcement, and infrastructure functions in the assurance boundary according to their actual contribution. Protect policy and supporting data as well as executable code.
5. Place enforcement where it can govern the consequential effect. Bind decisions to the operation, parameters, target, and conditions that are actually used.
6. *Engineering for Compromise* tests whether protection survives the authority of hostile requesting system elements. State explicitly which mechanisms, platforms, data sources, and administrators remain trusted.
7. Analyze unintentional adversity and self-induced emergent, erroneous, unsafe, or non-secure control actions. Rejection, delay, overload, and protective fallback can create loss as well as prevent it.
8. Compose the principle with *Mediated Access*, *Domain Separation*, *Least Privilege*, *Least Functionality*, *Least Sharing*, *Continuous Protection*, *Protective Failure*, and *Protective Recovery*.
9. Preserve the control properties through startup, maintenance, emergency operation, update, recovery, and disposal. Authorized changes can alter the assurance basis even when they preserve the visible policy wording.
10. Assurance must connect adequate constraints to dependable implementation under explicit assumptions. Tests, analysis, and mission validation support complementary parts of that argument.

© 2026 RONROSSECURE, LLC | CC BY 4.0.

12. Conclusion

Trustworthy System Control addresses a fundamental engineering question: why should a system's protective decisions be expected to constrain its behavior under adversity? A requirement or policy can express the intended restriction, but the realized mechanism must apply it wherever and whenever needed. Its own implementation, authority, supporting data, and dependencies determine whether that expectation is justified.

NIST provides a direct design assurance basis through the generalized reference monitor and its four essential criteria [1]. Non-bypassability addresses circumvention. An evaluatable mechanism supports rigorous assessment. A mechanism that is always invoked sustains protection whenever the constraint applies. Tamper protection preserves the functions and information on which the restriction depends. These criteria should be applied to the complete control function and the way its mechanisms compose.

Engineering for Compromise makes the distinction consequential. A hostile system element will use whatever authority and interfaces remain available. Protection should not rely on its voluntary cooperation. The architecture must identify the mechanisms that remain dependable, demonstrate the restrictions they enforce, and disclose the compromises that invalidate the claim [2]. The same discipline helps address faults, uncertainty, and harmful interactions among legitimate control functions.

Producers should make the control architecture and its assurance basis visible. Consumers should ask which effects are constrained, where enforcement occurs, who can change it, and what happens during failure and recovery. An identity check, signed update, or configured policy is useful evidence about a particular property. The system-level argument must connect those properties to the required mission behavior.

A trustworthy secure system depends on constraints that are both adequate for the loss concern and dependably enforced. Applying *Trustworthy System Control* turns that expectation into an engineering obligation that can be examined, substantiated, and sustained throughout the system life cycle.

"Specify what must remain true. Engineer the mechanisms that preserve it. Substantiate the result."

Acknowledgments

The author gratefully acknowledges the support provided by Dartmouth College's Institute for Security, Technology, and Society (ISTS). The resources provided by the ISTS contributed to the research and development of this paper. The views and conclusions expressed herein are those of the author and do not necessarily represent the views of Dartmouth College.

References

[1] R. Ross, M. Winstead, and M. McEvilley, Engineering Trustworthy Secure Systems, NIST Special Publication 800-160, Volume 1, Revision 1, National Institute of Standards and Technology, November 2022. https://doi.org/10.6028/NIST.SP.800-160v1r1

[2] R. Ross, "Engineering for Compromise: Designing Systems That Remain Trustworthy Secure Under Adversity," RONROSSECURE, LLC, August 2026. https://ronrossecure.com/library

[3] R. Ross, V. Pillitteri, R. Graubart, D. Bodeau, and R. McQuaid, Developing Cyber-Resilient Systems: A Systems Security Engineering Approach, NIST Special Publication 800-160, Volume 2, Revision 1, National Institute of Standards and Technology, December 2021. https://doi.org/10.6028/NIST.SP.800-160v2r1

© 2026 RONROSSECURE, LLC | CC BY 4.0.