

Engineering Mission Resilience in the Age of Frontier AI

Why Cybersecurity Must Operate Within a Systems Approach

Dr. Ron Ross

CEO, RONROSSECURE, LLC
ISTS FELLOW, DARTMOUTH COLLEGE

Abstract

Mission-critical systems fail at the seams between system components, teams, and authorities. A siloed cybersecurity approach can improve individual defenses while leaving the relationships among system components, authorities, services, and physical processes insufficiently constrained. Frontier AI widens those seams. It gives adversaries faster ways to explore and exploit a system, and it gives defenders new components that act with delegated authority and whose behavior cannot be fully specified. Faster detection does not fix excessive authority, a bypassable boundary, or a shared dependency that defeats recovery. Drawing on NIST SP 800-160, Vols. 1 and 2 (Revision 1) and the author's Engineering for Compromise methodology, this paper presents a loss-driven systems security engineering approach built on five commitments: (1) start from mission losses rather than the tool inventory; (2) assume selected system elements are compromised and ask what each element could cause; (3) constrain consequences outside the compromised element; (4) engineer recovery before the incident; and (5) keep the evidence for every protection claim and reevaluate it when the system changes. Traditional cybersecurity approaches and capabilities remain essential. Their contribution becomes more effective when coordinated around system-level protection needs and mission outcomes.

1. The Limits of Siloed Cybersecurity

An organization can build and operate a sophisticated security program and still depend on a fragile system architecture. Endpoints may be hardened, networks monitored, identities authenticated, and vulnerabilities tracked. Yet one compromised management service may control the software deployed across every mission domain. One broadly privileged account may alter operational data and recovery images. One shared dependency may disable both the primary service and its backup. These are system-level weaknesses that cannot be understood by examining individual defenses alone.

Public incidents illustrate the importance of system-level analysis. In the 2020 SolarWinds campaign, adversaries inserted malicious code into a trusted software update channel, allowing a compromised build and distribution path to reach thousands of customer environments [1]. In the 2015 attack on Ukrainian electricity distributors, intruders used remote-control tools and legitimate access paths to open breakers, interrupting service to roughly 225,000 customers [2]. In the 2021 Colonial Pipeline incident, ransomware affected business IT systems, and the operator halted pipeline operations for several days as a precaution [3]. Control systems did not have to be compromised for the mission to stop. These incidents illustrate why authority, shared dependencies, and containment require system-level analysis. The summaries alone do not establish that architectural deficiencies were the sole cause of each outcome.

“Organizational boundaries become analytical boundaries even though adversary influence crosses them.”

The problem is not specialization. Cybersecurity requires deep expertise in cryptography, software security, identity, network defense, incident response, and other disciplines. The problem arises when specialization becomes isolation: each team optimizes its assigned area without a common account of what the mission requires, which losses are unacceptable, and how the architecture can produce those losses.

A stovepipe perspective often treats security as a collection of products, security controls, and operational tasks applied to an existing system. A systems perspective examines how security requirements shape the system itself. It asks who can cause a consequential action, what dependencies make that action possible, which mechanisms constrain it, and whether those mechanisms remain dependable when other elements fail or become hostile. Both perspectives can use the same cybersecurity tools. They differ in the scope and purpose of the decisions those tools support.

NIST SP 800-160, Volume 1, Revision 1 provides the systems engineering foundation for this broader perspective. It treats security as an emergent property of an engineered system and integrates security considerations with systems engineering throughout the system life cycle [4]. Security depends on the composition of system elements and their interactions, not simply on the presence of security features within each element. A completed control assessment or a reduction in open vulnerabilities can provide useful evidence, but neither independently establishes mission resilience. The organization must also understand whether the realized system can continue to protect the mission under the adverse conditions included in its protection claims.

2. What Frontier AI Changes, and What It Does Not

For this paper, frontier AI refers to advanced general-purpose models and associated agent capabilities that can support complex reasoning, software tasks, and tool use. Security implications depend on the model, its deployment, available tools, assigned authority, and operational context. This paper does not assume that every model can autonomously compromise every system, or that defenders obtain no benefit from AI. Most of the design principles in this paper apply to any highly privileged automated system. A conventional management server with broad write access poses the same architectural problem as an AI agent with the same access. Frontier AI does not change the principles. It changes the circumstances in which they must be applied.

Tool-using AI agents change the architectural problem in several ways. Conventional automation typically separates code and data, although interpretation flaws remain possible; an AI agent can process natural-language content that influences both task selection and the information used to perform a task. Content processing must therefore never confer authority, and generated requests must be treated as untrusted input at consequential interfaces. Learned, open-ended behavior complicates specification and testing, increasing the importance of assurance focused on enforcement outside the model.

Credentials and tools bound an agent's authority, but model-selected actions can vary, requiring constraints on every direct and indirect consequential operation rather than only the intended task. AI models, prompts, memory, retrieval sources, and tools also introduce additional paths for behavior change, so configuration control must cover these artifacts and changes must trigger reassessment. Fluent, plausible outputs can obscure incorrect behavior, requiring mediators to evaluate the effects of requests as well as their form. Finally, both conventional automation and AI agents can operate at machine speed and scale, but flexible task selection can expand the paths an agent explores. Protection that depends on human response therefore requires explicit timing assumptions and a dependable alternative when intervention cannot occur in time.

On the adversary side, AI developers and threat intelligence teams have published reporting on attempts to use models for reconnaissance, code analysis, scripting, and social engineering [5][6]. The degree of automation and effectiveness varies and should not be assumed uniform. System architects should nevertheless examine scenarios in which an adversary explores available paths more quickly than people can investigate alerts, approve changes, or deploy repairs.

On the system defender side, tool integration creates interfaces; machine identities convey privileges; memory and retrieval services introduce data dependencies; and model and software updates alter behavior. Much of the security risk lies in legitimate capability used for an illegitimate purpose. A compromised agent does not need a new privilege-escalation exploit if its existing credentials already permit mission-critical changes. Prompt injection, which embeds hostile instructions in material an agent processes, is recognized as a leading risk class for model-based applications [7][8], and it becomes consequential when the surrounding architecture lets model-generated requests reach sensitive operations without independent constraints. A model's instruction to act safely does not establish that the system prevents an unsafe action.

AI also helps defenders with analysis, detection, testing, and engineering work, and those benefits should be pursued. But improved detection speed does not resolve excessive authority, a bypassable boundary, or a common dependency that defeats recovery. Because model behavior cannot be fully assured, enforcement outside the model moves from a supplementary control to the primary strategy. This is the intent of the *Assurance-bounded AI Component* principle proposed in earlier work: an AI component is never required to be more trustworthy than the system architecture can independently enforce [9]. The security objective is to reduce unnecessary capability and structurally limit the consequences of necessary capability, so that protection remains effective during the interval when detection or response is not.

“Because model behavior cannot be fully assured, enforcement outside the model moves from a supplementary control to the primary strategy.”

3. Begin with Mission Losses

A loss-driven approach starts with what stakeholders need the system to accomplish and the consequences they cannot tolerate. Threat analysis remains important, but the mission provides the basis for prioritizing it. This starting point is shared with loss-driven hazard analysis methods such as STPA [10]. An organization should be able to explain why a particular asset, function, interface, or dependency requires protection before deciding which mechanism to acquire or deploy.

Unacceptable losses can include death or injury, loss of essential infrastructure service, unauthorized disclosure of strategically valuable information, corruption of mission decisions, destruction of critical equipment, or loss of an operational opportunity that cannot be recovered. Their significance depends on context. A short interruption may be tolerable for an administrative application and unacceptable during a critical control sequence.

SP 800-160, Volume 1 connects security to *asset loss* and its unacceptable consequences [4]. Applying that foundation requires stakeholders to identify loss criteria, including relevant duration, scope, timing, and operating conditions. Engineers then examine the system conditions and interactions that could produce those consequences, whether from malicious action, defects, operator error, physical damage, or combinations of adverse events. The progression used in this paper can be expressed as:

Mission and Assets → Unacceptable Losses → Loss-Enabling System Conditions → Security Constraints → Architectural Allocation → Implementation → Evidence and Claims.

Each protection decision traces back to a mission loss and forward to evidence, and any system change reopens the affected steps for reassessment. The progression is an organizing method used in this paper, rather than a separate process prescribed by NIST. For example, a requirement to prevent unauthorized alteration of a safety-critical setpoint is incomplete if it addresses only the operator interface. Systems engineers must examine remote management, software updates, diagnostic access, recovery operations, and any AI tool capable of writing the same state. The constraint applies to the consequential effect wherever that effect can originate.

Loss analysis also prevents indiscriminate restriction. Eliminating a recovery function may reduce exposure while increasing the duration of a mission outage. Denying an emergency operator necessary authority may create a different unacceptable loss. The engineering task is to establish sufficient capability with bounded effects, and to make the tradeoffs visible to the stakeholders responsible for accepting them.

4. Engineer for Compromise

Engineering for Compromise (EfC), developed in earlier work [11], assumes selected elements may become untrustworthy. It asks: If an adversary controlled this system element, what could it cause the rest of the system to do? The answer must include misuse of legitimate functions and privileges. System element compromise, system compromise, and mission loss are distinct: independent boundaries may contain a hostile system element, while excessive authority may permit immediate mission loss. EfC examines these relationships and engineers barriers where consequences require them. The earlier paper develops containment boundaries, propagation analysis, and compromise margin; this section applies the method to AI-enabled systems.

“If an adversary controlled this system element, what could it cause the rest of the system to do?”

The EfC methodology has five steps:

1. **Define the compromise model.** State which system element is assumed hostile and at what level: application, operating system, hypervisor, firmware, administrator, or supplier. A restriction enforced by the operating system may constrain a hostile application while offering no protection against a hostile kernel. State what remains *trustworthy* and why.
2. **Enumerate inherited capability.** Map what the system element can read, write, invoke, delegate, consume, and influence through shared services, in every mode: startup, maintenance, emergency access, degraded operation, recovery, and disposal. A boundary that disappears during recovery is not dependable for a mission that depends on recovery.
3. **Trace to losses.** Follow each capability toward the unacceptable losses from Section 3. Prioritize paths where one compromise can cause severe loss, defeat several defenses, or corrupt recovery.
4. **Place barriers by precedence.** Apply the *Security Design Order of Precedence* (Section 5): eliminate the capability, reduce it, add engineered mediation, add visibility, and only then rely on procedure.
5. **State a bounded claim and collect evidence.** For a stated compromise scenario, the system may preserve a defined minimum service, prevent prohibited physical actions, restrict the exposure of sensitive information, or recover within a specified interval. Where the evidence does not support a claim, the design or the claim must change.

Table 1. Compromise analysis for a tool-using AI agent

AI AGENT	IF CONTROLLED BY AN ADVERSARY	CONSTRAINT TO ENGINEER
Tool credentials and API keys	Every operation the credentials permit, regardless of the intended task.	Issue only credentials for proposing actions; route consequential operations through a mediator the agent cannot modify.
Retrieved documents, messages, tool results	Hostile instructions embedded in processed content.	Processing content confers no authority; generated requests are treated as untrusted at consequential interfaces.
Persistent memory and retrieval stores	Poisoned state that influences later sessions or other users.	Integrity protection, provenance, and expiry for memory; separate stores by trust level.
Prompts, tool definitions, model artifacts	Altered behavior without any code change.	Separate update authority; independent approval; baseline and integrity checks.
Shared compute, queues, quotas	Impairment of an essential function without altering any data.	Reserve capacity for essential control; isolate the agent's resource pool.
The agent's own availability	Mission service depends on the agent.	A defined fallback that delivers essential service with the agent removed.

EfC does not imply that compromise is harmless or that every system element can become hostile simultaneously without consequence. Some system elements remain essential to the protection claim, and their failure can invalidate it. The architecture should identify those dependencies, minimize unnecessary trust, and require assurance proportionate to the consequences of misplaced confidence. Mission resilience is credible only within the conditions the architecture and evidence actually address.

5. Apply Security Design Principles as Architectural Decisions

The design principles in SP 800-160, Volume 1 provide a disciplined basis for determining where capability belongs and how its effects are constrained [4]. Many trace to earlier foundational work on protection mechanisms [12][13]. Their value comes from coordinated application to protection needs rather than from treating each as an isolated checklist. Earlier work discusses these principles in greater depth [14]; this section emphasizes what they require in AI-enabled systems.

The *Security Design Order of Precedence* in SP 800-160, Volume 1 makes the order of protection decisions explicit [4]. Engineers first seek to eliminate the potential for loss through design selection. Where elimination is impractical, they alter the design to reduce loss potential. They then incorporate engineered features or devices, provide visibility and feedback, and use signage, procedures, training, and equipment for the remaining needs. Removing an unnecessary agent tool, for example, eliminates a capability path before engineers must depend on a filter to manage its use.

- *Domain Separation* partitions system elements with different protection needs. Shared administrators, management planes, storage, and recovery services can undermine apparent separation, so the claim must cover the actual paths through which one domain can affect another.
- *Least Functionality* and *Least Privilege* limit what each system element can do and the authority it holds. An AI planning element may need to propose a schedule without the authority to deploy

software, change protection thresholds, or erase records. Effective privilege includes indirect influence: permission to replace an executable or recovery image confers the authority of whatever later runs that artifact.

- *Mediated Access* controls necessary interactions, constraining the requested operation, target, parameters, timing, system state, and resource use. External enforcement can constrain the consequences of prompt injection without depending on the model's interpretation of instructions, but it cannot alone establish that an authorized transaction is mission-appropriate, that supporting telemetry is correct, or that permitted outputs cannot disclose sensitive information.
- *Distributed Privilege* requires coordinated action before selected consequential operations and works only with meaningful and verifiable separation of authority. Two approval accounts under one compromised administrative domain do not provide the independence suggested by two nominal approvers.
- *Protective Failure* and *Protective Recovery* address what happens when trust conditions cease to hold and how they are reestablished; *Compositional Trustworthiness* asks whether evidence for the parts and their interactions supports confidence in the integrated system.

The protection mechanisms themselves require scrutiny. SP 800-160, Volume 1 identifies four essential design criteria [4], which correspond to the classic *reference monitor* properties [12]: protection must be non-bypassable, evaluable, always invoked, and tamper-proof. Systems engineers must also analyze what happens when a protection decision or enforcement function is wrong, compromised, delayed, or unavailable.

“Adding a mediator is insufficient if its failure permits the unacceptable behavior it was intended to prevent.”

These principles carry tensions. Added mediators can increase complexity or latency. Separation can reduce operational flexibility. Independent recovery can increase cost. Engineering analysis must weigh those costs against the losses prevented and the assurance achievable. Naming the principles does not establish that a design satisfies them.

6. Connect Bounded Compromise to Mission Resilience

NIST SP 800-160, Volume 2 complements the security foundation with cyber resiliency engineering [15]. Its goals are *Anticipate*, *Withstand*, *Recover*, and *Adapt*, with objectives that include *Continue*, *Constrain*, *Reconstitute*, and *Understand*. These orient design toward essential mission functions, the limitation of damage, restoration of trustworthy capability, and awareness of mission dependencies.

Mission resilience means the ability to sustain or restore essential mission outcomes under defined adversity. Cyber resiliency contributes, but mission resilience also depends on people, procedures, facilities, energy, communications, suppliers, and other enabling resources. Loss of a supporting resource can defeat the mission even when the application software remains uncompromised.

The cyber resiliency techniques help realize the architecture. *Segmentation* supports separation by criticality and trustworthiness. *Privilege Restriction* constrains authority under changing conditions. *Redundancy* provides protected instances of critical resources, while *Diversity* reduces susceptibility to common-mode failure. *Substantiated Integrity* helps determine whether resources have been corrupted. *Adaptive Response* implements courses of action when the current configuration is no longer suitable.

Selection requires dependency analysis. A standby service controlled by the same compromised deployment pipeline, credentials, or retrieval store may reproduce the primary service's failure. Two different models exposed to the same hostile input do not establish independent enforcement. Revoking an agent's credentials or isolating its domain preserves the mission only if a dependable fallback has been defined and can operate without those dependencies.

“Several copies of corrupted memory, configuration, or operational state may provide availability without trustworthy recovery.”

Recovery should be engineered before the incident. Protected restoration artifacts, separated recovery authority, procedures for reestablishing credentials, and checks on operational system state can all be necessary. Cryptographic authenticity does not prove that an approved image is free of defects or that restored data are correct; evidence must address suitability for the mission as well as its origin and integrity. For some missions the appropriate resilient behavior is continued operation with reduced capability. For others it is a controlled transition to a protective state. Maintaining an unsafe function continuously can be a failure of mission protection rather than a resilience success.

7. An Illustrative AI-Enabled Infrastructure System

Consider a hypothetical water utility that introduces an AI agent to optimize pumping schedules. The agent reads demand forecasts and operational telemetry and proposes changes. The mission is to maintain adequate water supply while avoiding hazardous pressure conditions and protecting critical equipment. This is an architectural illustration, not a report of a deployed utility or a claim of demonstrated performance.

In a siloed implementation, the AI team measures planning quality, the identity team provisions a service account, the operations team configures the controller connection, and the security team monitors the network. Each decision may appear reasonable locally. Together they may give the agent write access to operating setpoints, controller configuration, and a shared maintenance service. Hostile input or agent compromise could then turn an optimization function into a pathway to physical loss, much as legitimate remote-control tools became the attack path in Ukraine in 2015, except that in this case, the path is automated and human intervention may not occur before the hazardous action.

A systems approach first identifies unacceptable outcomes and the conditions that produce them. It determines that an optimization proposal must not override independently enforced operating limits, change protective settings, or disable emergency control, and it specifies the service that must continue if optimization becomes unavailable. System operators and safety engineers participate because those requirements cannot be derived from cybersecurity expertise alone.

The architecture separates the planning agent from the operational command path and allocates only the tools and privileges required for proposing schedules. A protected mediator translates approved proposals into a limited command vocabulary and evaluates them against authorized operating conditions. Independent protective mechanisms constrain physical behavior. Maintenance and recovery authority reside outside the agent's domain.

The engineering record must still address hard questions. Can compromised telemetry make a formally valid proposal dangerous? Can the agent exhaust a resource shared with essential control? Can an administrator bypass the mediator? Can the same update compromise planning and protection? If the mediator approves an invalid command, what independent limit prevents hazardous pressure? If recovery restores suspect software or data, what prevents premature return to service? Each answer

identifies a protection dependency that needs evidence or an architectural change; none can rest solely on the agent's expected behavior.

Table 2. How mission requirements change local decisions

DECISION	LOCAL SECURITY PERSPECTIVE	MISSION ENGINEERING REQUIREMENT
Agent identity	Authenticate the service account.	Bound every direct and indirect consequential operation.
Controller interface	Permit an encrypted connection.	Mediate commands and preserve independent protective limits.
Backup service	Maintain another available instance.	Prevent shared compromise and validate restoration state.
Incident response	Detect and disable the agent.	Preserve defined essential control without the agent.

Verification would attempt forbidden operations, bypasses, resource exhaustion, and restoration from suspect state, including hostile instructions in retrieved material and tool responses, poisoned memory reused across sessions, and quota exhaustion affecting essential control. Validation would examine whether the utility can sustain the required service and protective behavior when planning is unavailable or hostile. The resulting claim is specific: within stated assumptions, loss of trust in the optimization agent does not confer the authority needed to produce the identified unacceptable outcomes. The remaining risks and unsupported scenarios stay visible.

8. How This Differs from Other Approaches

Established security practices address many of these concerns. This approach integrates and extends their contributions around mission losses, explicit compromise models, architectural constraints, and evidence.

- *Defense in depth* provides layered controls so that one failure is not fatal; this approach makes layer independence explicit through shared-dependency analysis and connects each layer to the losses it constrains.
- *Zero trust* contributes per-request identity and access management, authentication and authorization, removal of implicit network trust, and protection of the policy decision process [16]; this approach links authorization policy to mission-loss criteria and examines authorized actions during normal, degraded, and recovery modes.
- *Threat-led testing* and ATT&CK-style modeling provide realistic adversary behavior and detection coverage [17]; this approach combines those capabilities with mission-loss prioritization and tests the misuse of legitimate capabilities.
- *STPA* and *STPA-Sec* provide systematic, loss-driven analysis of unsafe and insecure control actions [10][18]; this approach connects that analysis with explicit compromise models, assurance of protection mechanisms, and recovery.
- *AI risk management frameworks* contribute governance and life-cycle risk practices [19], complemented here by constraints designed to remain effective for any model behavior within stated compromise and protection assumptions.

- *Control-based compliance* provides baseline evidence of control presence, which this approach supplements with evidence of system behavior under defined compromise conditions.

9. Make Assurance a Continuing Engineering Responsibility

A systems approach needs evidence that connects stakeholder protection needs to the implemented system. Requirements, architecture descriptions, dependency models, interface specifications, privilege allocations, and configuration baselines establish what is intended. Analysis and testing establish whether those intentions are realized under the relevant operating and compromise conditions.

EfC does not remove the need for trustworthy components; it makes the placement of that trust explicit. A system may tolerate a hostile planning agent because independently protected elements constrain its effects, and those boundary enforcers, mediators, protective controllers, and recovery mechanisms must possess the properties on which the containment claim depends. The design principle of *Minimal Trusted Elements* keeps that dependency set small, the principle of *Commensurate Trustworthiness* relates required assurance and confidence to the consequences of failure, and the principles of *Substantiated Trustworthiness* and *Compositional Trustworthiness* require evidence for the individual system elements and a separate argument for the integrated result [4].

Common Criteria evaluation under ISO/IEC 15408 can supply bounded evidence about relevant IT components. It must be assessed against the evaluated version, configuration, claims, and operating assumptions, and it does not automatically transfer assurance to the composition. For an AI-enabled system, this permits a substantiated claim about an external enforcement boundary without requiring confidence that the model behaves correctly in every situation, provided the boundary addresses every consequential path within the claim's scope.

Measuring protection outcomes is also an important aspect of the EfC approach. Mission-based measures linked to protection outcomes can supplement vulnerability and incident counts. The following are proposed starting measures, to be calibrated during a pilot.

- **Loss-scenario coverage:** Share of identified unacceptable-loss scenarios with at least one independently enforced barrier verified by testing.
- **Single system element authority count:** Number of identities or system elements that can individually cause an unacceptable loss. The target is zero, or a formal justification for each exception.
- **Shared-dependency count:** Dependencies shared between a primary function and its fallback or recovery path.
- **Demonstrated restoration time:** Measured time to restore essential service from protected artifacts during an exercise, rather than an estimate.
- **Fallback exercise currency:** Time since essential service was last delivered successfully without the agent or primary system.
- **Claim currency:** Share of protection claims supported by evidence that remains current after the latest relevant system change.
- **Human-timing exposure:** Number of protective mechanisms that depend on human response within a stated interval, and how many have a tested alternative.

Assurance should concentrate on claims whose failure could cause unacceptable loss. Testing supports confidence within its scope; a successful exercise does not prove that all possible attacks are contained. The evidence must evolve with the system: a new agent tool, prompt configuration, memory policy,

supplier interface, model deployment, maintenance procedure, or recovery artifact can change what a system element can accomplish, and such changes should trigger review of affected claims.

Decision authority must cross functional boundaries. Mission owners establish priorities and acceptable loss criteria and accept residual risk within their authority. The system architect owns the integrated dependency model and the architectural decisions. Systems security engineers develop protection constraints and assurance claims. The Chief Information Security Officer (CISO) and security operations team contribute threat knowledge, detection, and response. Safety specialists, operators, suppliers, and recovery teams establish conditions for protective operation and restoration. Unresolved tradeoffs go to the authority responsible for the affected mission outcome.

10. Cost, Feasibility, and Legacy Systems

Senior leaders will ask what this approach costs to implement—a reasonable question. The honest answer is that costs are organization-specific and should be measured in a pilot, but their sources are predictable. Typically, costs arise from a cross-functional analysis effort, architectural changes such as mediators, separated administration, independent recovery, and reserved capacity; operational friction from approvals, latency, and reduced flexibility; and recurring assurance as the system changes. Reusing artifacts may reduce later analysis effort, but the pilot should measure that benefit.

There are also potential cost savings. Removing unnecessary capability may reduce protection and operating costs. Bounded consequences may reduce incident losses and dependence on rapid response. Resolving protection needs during design or acquisition may avoid expensive retrofits. Evaluate these benefits against implementation, transition, and assurance costs in the pilot.

For legacy systems, a complete redesign is rarely feasible. Staged improvements can place a mediator in front of an unmodifiable legacy interface, add independent physical or hardware limits, separate update and recovery authority, and remove functions before adding controls. The design principle of *Commensurate Trustworthiness* directs the heaviest assurance spending to the few elements whose failure would be most impactful.

When a specific constraint cannot be achieved, the options are explicit: withhold the authority; add a compensating independent barrier; defer the deployment; or have the accountable mission owner formally accept the residual risk, with a stated scope and expiry date. What the approach does not permit is silent acceptance by default.

There will also be potential organizational obstacles. The usual obstacles are unclear ownership of the integrated architecture, incentives that reward feature delivery over constrained design, and security reviews that occur after key decisions are made. Remedies are typically structural in nature: link the analysis to acquisition, architecture review, release approval, and supplier agreements, and give the system architect and mission owner explicit authority and executive sponsorship.

11. Putting the Approach into Practice

Begin with a focused pilot involving one essential mission function. Use it to demonstrate how explicit authority limits, independently enforced constraints, dependable fallback, and protected recovery can prevent or limit unacceptable mission losses.

Table 3. Illustrative 90-day pilot with exit criteria

PHASE	ACTIVITY	EXIT CRITERIA
Days 1 to 30	Establish the mission boundary, loss criteria, minimum acceptable service, and decision authorities. Map consequential interfaces, privileges, shared dependencies, and recovery resources. Involve operations and safety staff from the start.	Loss criteria approved by the mission owner. Dependency map reviewed by operations and safety. Named architect and sponsor.
Days 31 to 60	Assume selected exposed or powerful elements are hostile. Trace what they can read, alter, invoke, delegate, exhaust, or disable toward unacceptable outcomes. Test the credibility of human-response timing assumptions. Select a small set of architectural improvements and evaluate their effects on performance, safety, maintainability, cost, and assurance.	Written compromise models for the highest-authority elements. Ranked list of single-element paths to severe loss. Selected improvements with acceptance criteria.
Days 61 to 90	Implement the highest-priority feasible improvements, such as removing an unnecessary tool, narrowing a service identity, separating update authority, independently mediating a consequential interface, or protecting recovery capability. Demonstrate essential service during isolation, forbidden operations under compromise, independence of critical dependencies, and conditions for accepting restored capability.	Each targeted path eliminated, independently barred, or formally accepted. Fallback and restoration demonstrated. Evidence, unsupported scenarios, and residual risk presented to the mission owner.

The following leadership decisions establish the pilot's scope, accountability, and resources.

- Name the three most unacceptable losses for one essential mission function.
- Assign one accountable architect to own the integrated dependency model.
- Require every tool-using AI agent to have documented authority limits enforced outside the model.
- Require a tested fallback that delivers essential service without the agent.
- Fund recovery capabilities whose credentials, pipelines, and restoration stores are protected independently from the primary system's identified compromise paths.

Common failure modes that can occur in a pilot include:

- **Scope too broad:** Keep the pilot focused on one function and expand only after evidence supports the initial protection claims.
- **Operations and safety engaged late:** Involve these personnel from the outset so missed requirements do not emerge as outages or unsafe designs.
- **Findings without authority to act:** Escalate decisions beyond the pilot team's authority before the pilot ends.
- **Out-of-date evidence:** Reopen and reassess affected protection claims when system changes alter their supporting assumptions or mechanisms.

- **A system element discovered that cannot be constrained:** Treat the finding as a useful engineering result and evaluate the options discussed in Section 10.

The pilot should produce evidence supporting the protection claims and identify necessary architectural changes. It should not imply that the entire system's mission resilience has been established.

12. Engineer Security Around Mission Outcomes

Frontier AI adds urgency to weaknesses created by excessive capability, concentrated authority, uncontrolled interaction, and fragile dependencies. Those weaknesses demand decisions about the system's structure as well as its operational defenses. Faster tools can help, but trustworthy secure systems require a defensible account of what the architecture permits under *adversity*.

A loss-driven approach establishes what must be protected and why. *Engineering for Compromise* tests whether selected losses of trust can propagate into unacceptable consequences. The security design principles guide the allocation of capability and authority, and cyber resiliency engineering supports continued operation, adaptation, and trustworthy recovery.

“Evidence determines the confidence justified in the resulting claims.”

The challenge is organizational as well as technical. Cybersecurity experts must participate in the engineering decisions that create the system capability and architecture. Mission leaders must take responsibility for the system's protection needs and tradeoffs. The goal is a system whose behavior under compromise has been deliberately constrained, whose essential outcomes remain achievable within defined limits, and whose *resilience* is supported by evidence.

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. The AI scenarios, examples, and architectural applications in this paper are the author's engineering synthesis, not findings attributed to the cited NIST publications.

References

- [1] CISA. Alert AA20-352A, “Advanced Persistent Threat Compromise of Government Agencies, Critical Infrastructure, and Private Sector Organizations.” December 2020. <https://www.cisa.gov/news-events/cybersecurity-advisories/aa20-352a>
- [2] E-ISAC and SANS ICS. “Analysis of the Cyber Attack on the Ukrainian Power Grid.” March 2016.
- [3] CISA and FBI. Alert AA21-131A, “DarkSide Ransomware: Best Practices for Preventing Business Disruption from Ransomware Attacks.” May 2021. <https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-131a>
- [4] Ross, R., Winstead, M., and McEvilly, M. *Engineering Trustworthy Secure Systems*. NIST SP 800-160, Volume 1, Revision 1. November 2022. <https://doi.org/10.6028/NIST.SP.800-160v1r1>
- [5] Google Threat Intelligence Group. “Adversarial Misuse of Generative AI.” January 2025.
- [6] OpenAI. “Disrupting Malicious Uses of AI.” June 2025.

- [7] OWASP. *Top 10 for Large Language Model Applications*, 2025 edition. November 2024. <https://genai.owasp.org/llm-top-10>
- [8] MITRE. *ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems*. <https://atlas.mitre.org>
- [9] Ross, R. “A Multidimensional Protection Strategy for Engineering Mission-Resilient Systems in the Age of Frontier AI.” RONROSSECURE, LLC, July 2026. Available through the online library at <https://ronrossecure.com>.
- [10] Leveson, N. G. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2011.
- [11] Ross, R. Engineering for Compromise: Designing Systems That Remain Trustworthy Secure Under Adversity. RONROSSECURE, LLC, August 2026. Available through the online library at <https://ronrossecure.com>.
- [12] Anderson, J. P. *Computer Security Technology Planning Study*. ESD-TR-73-51, U.S. Air Force Electronic Systems Division. 1972.
- [13] Saltzer, J. H., and Schroeder, M. D. “The Protection of Information in Computer Systems.” *Proceedings of the IEEE* 63(9), 1975, pp. 1278–1308.
- [14] Ross, R. “Trustworthy Secure Design: Engineering Security from the Inside Out.” RONROSSECURE, LLC, August 2026. Available through the online library at <https://ronrossecure.com>.
- [15] Ross, R., Pillitteri, V., Graubart, R., Bodeau, D., and McQuaid, R. *Developing Cyber-Resilient Systems: A Systems Security Engineering Approach*. NIST SP 800-160, Volume 2, Revision 1. December 2021. <https://doi.org/10.6028/NIST.SP.800-160v2r1>
- [16] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. *Zero Trust Architecture*. NIST SP 800-207. August 2020. <https://doi.org/10.6028/NIST.SP.800-207>
- [17] MITRE. ATT&CK. <https://attack.mitre.org>
- [18] Young, W., and Leveson, N. G. “An Integrated Approach to Safety and Security Based on Systems Theory.” *Communications of the ACM* 57(2), 2014, pp. 31–35.
- [19] National Institute of Standards and Technology. *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. NIST AI 100-1. January 2023. <https://doi.org/10.6028/NIST.AI.100-1>