Engineering for Compromise

Designing Systems That Remain Trustworthy Secure Under Adversity

Dr. Ron Ross

CEO, RONROSSECURE, LLC.
ISTS FELLOW, DARTMOUTH COLLEGE

Abstract

Modern systems cannot credibly depend on every component, account, service, person, supplier, and supporting dependency remaining trustworthy secure throughout the system life cycle. Prevention-centered cybersecurity remains essential, but for mission-critical systems it is not a sufficient engineering strategy. This paper introduces the concept of Engineering for Compromise (EfC), a systems security engineering strategy that deliberately assumes selected system elements may fail, become malicious, lose integrity, or fall under adversary control and then asks whether the system can continue to prevent unacceptable loss. EfC separates component compromise, system compromise, and mission loss as distinct events and uses architecture to place barriers between them. The approach is grounded in the loss-driven systems security engineering concepts and security design principles of NIST Special Publication 800-160, Volume 1, Revision 1. Domain Separation, Least Privilege, Least Sharing, Least Functionality, Mediated Access, Distributed Privilege, Protective Failure, Anomaly Detection, Protective Recovery, and Compositional Trustworthiness combine to bound adversary authority, limit propagation, preserve critical system properties, and restore an authenticated trustworthy secure state. The paper develops the concepts of compromise containment boundaries, compromise propagation analysis, and compromise margin; describes the evidence needed to substantiate containment claims; and applies the approach across the complete system life cycle. The central proposition is simple: a trustworthy secure system is not necessarily one in which nothing can be compromised. It is one engineered so that compromise does not automatically become catastrophe.

1. Introduction

The cybersecurity community has spent decades improving its ability to prevent compromise. That work is indispensable. Authentication, access control, vulnerability management, encryption, endpoint protection, network segmentation, monitoring, threat intelligence, supply-chain controls, and incident response all reduce the probability that an adversary can establish a foothold in an organizational system. Yet the architecture of a *mission-critical* system cannot rest on an assumption that every preventive measure will always succeed. Modern systems are too interconnected, software-intensive, externally dependent, and continuously changing for universal and permanent component trustworthiness to be a credible design premise.

The more useful engineering question is not whether compromise can be made impossible, but **what the system permits an adversary to accomplish after compromise occurs**. A compromised administrator account, endpoint, network device, processor, cloud service, sensor, supplier component, or maintenance tool is an adversary condition. It is not automatically a *mission* loss. Whether that condition becomes catastrophic depends on the structure of the system: the authority assigned to the compromised element, the information it can reach, the interfaces it can invoke, the dependencies it can influence, the boundaries it can cross, and the protective responses available when assumptions begin to fail.

Engineering for Compromise (EfC) begins with that distinction. It treats *component compromise*, *system compromise*, and *unacceptable loss* as separate events and deliberately engineers barriers between them. Prevention remains the first opportunity to stop adversity. EfC addresses the next question: if prevention fails locally, can the system keep that failure local? The goal is not to make compromise acceptable. **The goal is to deny the adversary the assumption that one successful penetration must become a successful mission attack**.

This perspective is consistent with NIST SP 800-160, Volume 1, Revision 1, which treats security as an emergent system property and applies systems security engineering across the system life cycle [1]. The publication's design principles provide a vocabulary for structuring trust, authority, information flow, failure, recovery, and evidence. EfC composes those principles around a specific engineering objective: preserving critical system-level properties under conditions in which selected elements can no longer be trusted. The companion analysis of RAND's Secure Inference Data Center provides a concrete high-assurance example of this broader systems-security argument [2].

The approach is intentionally technology-neutral. It applies to national-security platforms, critical infrastructure, industrial control systems, cloud and distributed systems, financial infrastructure, communications, transportation, space systems, enterprise systems, and artificial intelligence. The details of compromise differ across domains, but the governing question is the same: what unacceptable losses become reachable when an element is assumed hostile, and what architectural constraints prevent that reachability? Unlike conventional *assume-breach* strategies, EfC converts compromise assumptions into (1) loss-driven architectural constraints, (2) compromise containment boundaries, (3) propagation analysis, (4) compromise margins, and (5) substantiating evidence.

> ***A trustworthy secure system should not have to depend on every component remaining trustworthy for the system itself to remain trustworthy.***

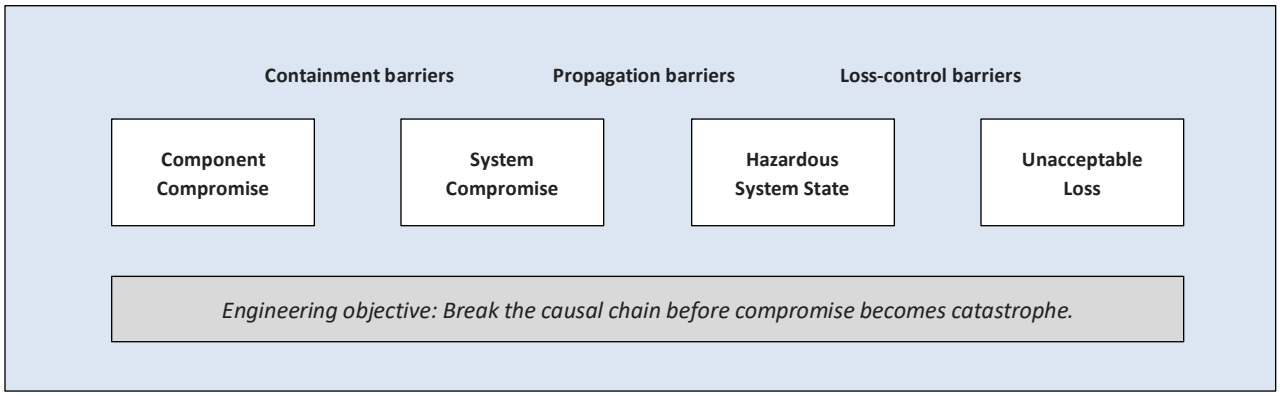

Figure 1. *Engineering for compromise containment, propagation, and loss-control barriers*

2. Change the Security Objective

Cybersecurity and systems security engineering view the problem from complementary perspectives. Threat-centric cybersecurity commonly begins with an adversary, a vulnerability, an exploit path, and a *control* intended to prevent or detect exploitation. Loss-driven systems security engineering begins with the mission, the assets and capabilities stakeholders depend upon, the consequences that cannot be tolerated, and the system conditions that can lead to those consequences. The former asks how an attack may occur. The latter asks what the system must prevent regardless of the particular path by which adversity arises.

For EfC, this distinction is decisive. Compromise is an adversary condition; *loss* is a system consequence. A maliciously controlled component matters because of the system states it can create or influence. The engineering task is therefore to constrain the causal path from adversary foothold to unacceptable consequence. This reframes the security objective from "keep the adversary out everywhere" to "prevent unacceptable loss even when selected preventive assumptions fail."

A useful loss-driven chain is:

> **Mission → Unacceptable Loss → Hazardous System Condition → Security Constraint → Architecture → Mechanism → Evidence.**

Mission analysis identifies what must be protected and why. Unacceptable losses define the consequences that justify protection. Hazard analysis identifies system states or control actions that can produce those losses. Security constraints specify what must or must not occur. Architecture allocates trust, authority, information flow, and responsibility. Mechanisms realize the constraints. Evidence establishes justified confidence that the realized system preserves them.

This approach changes design conversations. Rather than asking only whether an endpoint is hardened, engineers ask whether compromise of that endpoint can authorize a consequential action. Rather than asking only whether a network is segmented, they ask whether any shared identity, management, storage, timing, physical, or administrative dependency defeats the intended separation. Rather than asking whether recovery restores service quickly, they ask whether the recovery basis itself is trustworthy. **The unit of analysis becomes the system and its mission, not the individual security product.**

EfC can therefore be defined as the deliberate application of systems security engineering principles to limit the authority, reachability, propagation, and consequences available to an adversary after one or more system elements have become untrustworthy. The definition is intentionally broader than cyber intrusion. A system element may become untrustworthy because of malicious compromise, latent defect, operator error or misuse, supply-chain manipulation, physical damage, configuration drift, or failure of an external dependency. If the resulting system condition is equivalent from the standpoint of unacceptable loss, the architecture should respond accordingly.

3. Assume the System Element Is Hostile

The most revealing EfC design question is intentionally severe: **if this element were completely controlled by the adversary, what damage or loss could it cause**? The analysis should not be limited to the behavior of a known exploit. It should assume the adversary can exercise every legitimate privilege, invoke every reachable interface, manipulate every writable object, exploit every permitted timing relationship, and misuse every dependency available to the element. The purpose is to expose the true architectural consequence of lost *trust*.

This leads to the concept of a *Compromise Containment Boundary (CCB)*: the architectural boundary intended to limit the information, authority, control, functionality, and influence available through one or more compromised system elements. A CCB is not necessarily a subnet, enclave, virtual machine, device, or physical room. It is a system-level construct that may combine cyber, physical, procedural, human, and organizational constraints.

For each assumed compromise, systems engineers should map the compromise reach: information that can be read or inferred; data that can be modified; privileges that can be exercised; identities that can be impersonated; functions that can be invoked; states that can be changed; physical processes that can be influenced; interfaces and maintenance paths that can be used; and other domains that can be reached

directly or through shared dependencies. The analysis should include exceptional modes such as startup, maintenance, degraded operation, emergency access, recovery, and disposal because containment that exists only during normal operation is not dependable containment.

The critical test is whether everything reachable from the compromised element is sufficient, alone or in combination with plausible additional failures, to cause an unacceptable loss. If the answer is yes, the architecture must change, the authority must be reduced, additional independent barriers must be introduced, or the element must be elevated into a smaller set of trusted elements that receive assurance commensurate with their consequence.

CCBs also expose an important truth about trust. Trust is not a label attached to a product. It is a dependency created by the architecture. An element is trusted for a claim when failure or malicious behavior of that element can invalidate the claim. EfC therefore seeks not universal trustworthiness but deliberate trust placement: **make the trusted set as small, simple, independent, and substantiated as practicable**.

4. Architect for Bounded Compromise

Bounded compromise is an emergent architectural property. No single control or design principle creates it. It results from composing principles that reduce what an adversary gains from compromise and constrain how that gain can propagate. NIST SP 800-160 provides a particularly useful set of security design principles for this purpose [1].

- *Domain Separation* establishes the containment structure. Trust, authority, information, control, and failure are partitioned so that compromise in one domain does not automatically confer capability in another. Separation should extend beyond network topology to identities, administrative authorities, management services, storage, physical access, recovery infrastructure, and enabling systems. A boundary that can be bypassed through a common administrator or shared writable service is not a meaningful compromise boundary.
- *Least Privilege* bounds post-compromise authority. The conventional question (i.e., what privileges does this component need?) should be supplemented by a hostile-use question: **if every assigned privilege were exercised maliciously, what losses become reachable**? Privilege is not merely an entitlement-management concern; it is latent adversary capability once the holder is compromised.
- *Least Sharing* reduces propagation infrastructure. Shared identity systems, management planes, hypervisors, libraries, writable storage, networks, support tools, and common personnel can create paths through which one compromise becomes many. Sharing is often operationally efficient, but every shared mechanism creates coupling. EfC makes the security cost of that coupling explicit and requires justification when common mechanisms cross important containment boundaries.
- *Least Functionality* denies capability to the future adversary. Unnecessary services, ports, protocols, interpreters, administrative interfaces, persistence mechanisms, diagnostic functions, and update paths may be harmless while the component is *trustworthy* but become useful tools after compromise. Minimality therefore has a direct compromise-containment purpose: **remove capability the mission does not need before an adversary can inherit it**.
- *Mediated Access* governs interactions that must remain. Isolation cannot be absolute in a useful system, so necessary crossings should be explicitly defined, authenticated, authorized, bounded, observable, failure-aware, and non-bypassable. A mediator should constrain vocabulary, direction,

capacity, timing, privilege, state transitions, and error behavior as appropriate to the loss being controlled.

- *Distributed Privilege* makes one compromise insufficient for selected consequential actions. Two-person control, threshold cryptography, independent administrative domains, separate approval chains, and split custody are examples of a broader principle: **do not concentrate all authority required for catastrophic action in one system element that can be compromised**. The design objective is not administrative formality; it is architectural resistance to unilateral adversary control.
- *Trustworthy System Control* ensures that the mechanisms enforcing containment cannot simply be disabled or bypassed by the elements they constrain. Together, these design principles produce a well-orchestrated progression:

Separate → Minimize → Limit → Mediate → Distribute

The result is not that compromise disappears, but that the capability acquired through compromise is *intentionally bounded*.

Table 1. NIST SP 800-160 security design principles as Engineering-for-Compromise mechanisms

DESIGN PRINCIPLE	EfC PURPOSE	COMPROMISE-CONTAINMENT EFFECT
Domain Separation	Partition trust, authority, information, control, and failure.	Compromise in one domain does not automatically confer capability in another.
Least Privilege	Limit authority available to each element.	A compromised element inherits only bounded authority.
Least Sharing	Reduce common services and writable dependencies.	Fewer shared mechanisms exist for compromise propagation.
Least Functionality	Remove unnecessary behavior and latent capability.	The adversary inherits less functionality after compromise.
Mediated Access	Constrain necessary interactions across boundaries.	Crossings are explicit, bounded, observable, and non-bypassable.
Distributed Privilege	Split authority required for consequential actions.	No single compromise is sufficient for selected catastrophic actions.
Protective Failure / Defaults	Move toward defined protective states under uncertainty.	Degradation or shutdown occurs before loss thresholds are crossed.
Anomaly Detection	Observe violations of architectural invariants.	Loss of containment assumptions is detected even without attacker attribution.
Protective Recovery	Restore an authenticated trustworthy basis.	Service is not resumed until trust conditions are reestablished.
Compositional Trustworthiness	Substantiate that local claims survive integration.	System-level containment is supported by evidence, not component pedigree.

5. Moving from Compromise to Catastrophe

EfC requires an explicit analysis of how local compromise could propagate toward loss. A *Compromise Propagation Analysis (CPA)* can complement conventional attack-path and threat analysis by focusing on system reachability after trust has been lost. The starting assumption is not “the adversary might exploit this component” but “this component is already hostile; what becomes reachable next?”

Propagation can occur through privilege, identity, information flow, control flow, software and firmware distribution, shared dependencies, physical processes, administrative authority, supply-chain relationships, and recovery mechanisms. These dimensions often interact. A compromised management service may provide credentials; those credentials may provide access to a shared storage service; that service may influence configuration in another domain. No individual step appears catastrophic, yet *composition* creates a path to loss.

The analysis should model a progression from initial compromise to additional capability, boundary crossing, privilege or control expansion, hazardous system state, and unacceptable loss. At each transition, engineers identify the independent barrier intended to interrupt the chain. The objective is not to count controls but to establish whether the barriers are genuinely independent, diverse where necessary, non-bypassable, and resistant to common-mode failure.

This is a more rigorous interpretation of defense in depth. Multiple mechanisms that all depend on the same identity provider, administrator, hypervisor, cryptographic root, or management network may appear layered while sharing a single point of failure. EfC treats independence as an architectural property that must be demonstrated, not assumed.

6. Addressing the Compromise Margin

NIST SP 800-160 includes the principle of *Loss Margins*, which emphasizes maintaining sufficient distance from conditions associated with unacceptable loss [1]. EfC extends that reasoning with an analytical concept that can be called *compromise margin*: **the amount of adversary success, loss of trust, or component compromise that can occur before an unacceptable-loss state becomes reachable**. Compromise margin is not proposed as a universal numerical metric; its value is in forcing explicit statements about what the architecture actually tolerates.

A system may be designed to tolerate one compromised endpoint but not a compromised domain controller; one malicious operator but not two cooperating operators; loss of one communications path but not the common timing service used by all redundant system paths. Those distinctions should be visible in the *security argument*.[1] If the system's survival depends on a particular element never being compromised, that dependency should be identified rather than hidden inside an aggregate risk score.

Useful categories include: (1) single-compromise tolerance, (2) multiple independent compromise tolerance, (3) correlated compromise, and (4) common-mode compromise. The analysis should also consider sequence. Two compromises that are harmless independently may become catastrophic if one creates the conditions for the second. Conversely, a protective state entered after the first compromise may intentionally reduce the system's attack surface and increase the margin against further adversary action.

Compromise margin provides a bridge between architecture and resilience. Resilience is not simply the ability to keep operating after adversity. For mission-critical systems, resilience includes the ability to preserve critical security properties, constrain damage, and recover an authorized state even when some mission capability must be sacrificed.

[1] A *security argument* is the structured reasoning that explains why a system should be considered adequately secure (e.g., how architectural constraints prevent specified unacceptable losses). An *assurance case* is the documented body of claims, arguments, and supporting evidence used to justify confidence in the system's trustworthiness. Thus, the security argument is part of the assurance case.

7. Protective Failure Sometimes Means the System Must Stop

Availability-oriented operational cultures can create pressure to continue service under uncertainty. EfC requires the opposite question when critical security properties are at risk: can continued operation occur without moving the system closer to unacceptable loss? If not, degradation, isolation, suspension, or shutdown may be the correct engineered response.

Protective Failure and *Protective Defaults* provide the conceptual basis. When authentication becomes uncertain, a mediator fails, telemetry is lost, configuration cannot be attested, or a critical boundary assumption is violated, the system should move toward a predefined protective state rather than improvising permissive behavior. Depending on the mission, that state may revoke authority, disable selected functions, isolate a domain, reduce connectivity, preserve evidence, or terminate processing.

The key is to design protective states in advance. A shutdown path that exposes sensitive state, a failover mechanism that bypasses mediation, or an emergency account with unlimited cross-domain privilege can transform a protective response into a new attack path. Protective failure must therefore be part of the architecture, requirements, verification, and exercises, not solely an incident-response capability.

A system that deliberately sacrifices functionality to prevent catastrophic loss is not necessarily failing. It may be demonstrating successful system security engineering. EfC makes that trade explicit and ties it to stakeholder-defined loss priorities.

8. Monitor the Security Argument, Not Just the Attacker

Traditional security monitoring frequently concentrates on recognizing malicious behavior: known indicators, anomalous processes, suspicious network traffic, or attacker techniques. EfC adds another monitoring objective: **determine whether the assumptions and invariants upon which the system's security claims depend are still true**.

Architectural invariants may include the absence of prohibited flows, continued operation of required mediators, bounded privilege, separation of administrative authorities, integrity of critical configuration, validity of cryptographic roots, independence of redundant mechanisms, and availability of the recovery basis. These properties are directly connected to the loss-control argument. Their violation matters even when the organization cannot identify the attacker or classify the technique.

This approach changes *Anomaly Detection* from a largely operational cybersecurity function into an engineering function. Requirements specify which invariants must be observable, architecture determines where evidence can be collected without creating new compromise paths, design establishes thresholds and response authority, and verification demonstrates that detection occurs with sufficient timeliness and confidence to prevent loss.

The result is monitoring of the security argument itself. The system is not merely asking whether an attack is visible. It is asking whether the conditions that justify continued trust remain valid.

9. Protective Recovery Requires Restoring Trustworthiness

Recovery is often measured by restoration of availability. For EfC, service restoration is necessary but not sufficient. A system that restarts with compromised keys, corrupted firmware, invalid identities, weakened boundaries, or unverified operator authority has restored function without restoring *trustworthiness*.

Protective Recovery therefore means reestablishing the conditions required to substantiate the system's security claims. Depending on the system, that may include hardware identity, firmware, and software integrity, cryptographic keys, credentials, configuration, authoritative data, privilege assignments, physical boundaries, monitoring, external dependencies, and personnel authority. Recovery should establish a known and authenticated basis rather than merely return the system to a previous image.

The recovery system itself must be *engineered for compromise*. Backup repositories, golden images, key vaults, recovery credentials, maintenance networks, and emergency procedures can become high-value adversary targets precisely because normal security boundaries may be relaxed during restoration. EfC therefore treats the recovery basis as a security-critical system with its own containment, separation, provenance, and evidence requirements.

Recovery is complete only when the conditions necessary to justify *trust* have been reestablished. That principle prevents availability metrics from masking persistent compromise.

10. Compositional Trustworthiness Requires Trusting Less

One of the hardest system security problems is composition. Individually trustworthy components do not necessarily compose into a trustworthy system because interactions create new behaviors, dependencies, timing relationships, privilege combinations, and failure paths. EfC adds the converse insight: **a trustworthy system need not require every component to remain highly trustworthy if the architecture can safely tolerate compromise of selected elements**.

This suggests a more productive question than “how can every component be made trustworthy?” Ask instead: “how few components must remain trustworthy for the system-level security claim to survive?” The answer identifies the *trusted footprint*—the system elements whose malicious failure could defeat containment or directly enable unacceptable loss.

The highest *assurance* effort should be concentrated on that footprint. Critical mediators, roots of trust, authorization mechanisms, recovery authorities, and selected physical or procedural controls should be made as few, small, simple, deterministic, isolated, and rigorously analyzed as practicable. Commodity or lower-assurance system components can then operate within an architecture that assumes their possible compromise.

Compositional Trustworthiness requires that assumptions be explicit at every boundary. A formally verified mediator is useful only if its implementation matches the model, its surrounding environment satisfies the proof assumptions, its control cannot be bypassed, and other system paths do not recreate the prohibited capability. Local evidence becomes system evidence only when the composition preserves the system-level constraint.

Trust minimization is therefore not pessimism about technology. It is disciplined *allocation* of assurance. The system should depend on strong trust only where the consequences demand it and should use architecture to keep that trusted set as small as possible.

11. Engineer for Multiple and Coordinated Compromise

Sophisticated adversaries do not have to respect architectural assumptions about one-at-a-time failure. Nation-state adversaries and similarly capable actors may coordinate cyber operations, insider access, physical intrusion, supply-chain manipulation, credential theft, service-provider compromise, and deception. EfC must therefore examine combinations of compromise and the possibility that apparently independent barriers share a common cause.

Common-mode compromise is especially dangerous because it creates false confidence in redundancy. Two authorization services are not meaningfully independent if both are administered from the same device or workstation. Two network paths are not independent if they share the same control plane. Two software implementations may share a vulnerable library or build pipeline. Separate operators may rely on the same identity provider or recovery credential.

The architecture should therefore identify independence assumptions explicitly and test them. *Diversity*, *Redundancy*, *Distributed Privilege*, *Domain Separation*, and *Self-Reliant Trustworthiness* can reduce correlated failure, but only when the underlying dependencies are genuinely distinct. Where independence cannot be achieved, the assurance argument must acknowledge the resulting concentration of risk.

Multiple-compromise analysis also supports realistic adversary modeling. Rather than assuming a sophisticated attacker or an unrealistically weak one, engineers can ask which combinations of footholds are credible for the stated adversary and which combinations the system must tolerate. This makes the required compromise margin a *mission decision* rather than an accidental property of implementation.

12. Evidence That Compromise Will Remain Bounded

EfC is incomplete without evidence. An architecture may claim that compromise of *Domain A* cannot produce unauthorized control of *Domain B* sufficient to cause a specified loss, but that claim requires objective substantiation. Security mechanisms provide protection; evidence provides justified confidence that the mechanisms and their composition produce the claimed system property.

Evidence should be selected according to the claim and consequence. Architectural and information-flow analysis can expose unintended paths. Privilege analysis can identify combinations of authority. Formal methods and model checking can establish selected properties of bounded protocols and state machines. Static analysis, simulation, fault injection, penetration testing, adversarial testing, red teaming, insider-path analysis, physical testing, and recovery exercises address different dimensions of the system.

No single method proves the whole system trustworthy secure. Formal proof of a boundary controller does not establish containment if maintenance equipment bypasses it. Red teaming can reveal weaknesses but cannot exhaust the state space. Penetration testing does not demonstrate absence of paths not exercised during the test. EfC therefore uses complementary evidence and keeps each claim carefully bounded to its assumptions.

The *assurance case* should trace unacceptable loss to hazardous states, constraints, requirements, architecture, mechanisms, implementation evidence, integration evidence, verification results, operational monitoring, and recovery exercises. It should also identify the assumptions that would invalidate the claim. Evidence has an operational lifetime: a configuration change, new dependency, changed mission, or revised adversary assumption may require the claim to be reevaluated.

The strongest EfC claim is therefore not “this component is secure.” It is a system-level statement such as “under the defined adversary and operating assumptions, compromise of the specified element or domain cannot produce the identified unacceptable loss without defeating the stated independent barriers.” That is a claim engineers can analyze, test, and maintain.

13. Engineering for Compromise Across the System Life Cycle

Compromise tolerance cannot be added during operations after the architecture has already concentrated authority and created shared dependencies. NIST SP 800-160 applies systems security engineering across all system life-cycle processes. EfC should be treated as a thread through all of them [1].

Business or Mission Analysis establishes which losses justify compromise tolerance and what mission tradeoffs are acceptable. *Stakeholder Needs and Requirements Definition* identifies which capabilities, assets, and outcomes stakeholders require to remain protected under adversity. *System Requirements Definition* converts those needs into explicit compromise-tolerance obligations, including required states, modes, transitions, and failure behavior.

Architecture Definition establishes containment boundaries and allocates trust, authority, information flow, and failure isolation. *Design Definition* selects the mechanisms that enforce those boundaries. *System Analysis* evaluates compromise propagation, combinations of failures, common-mode dependencies, compromise margin, and tradeoffs between availability and protective failure.

Implementation must preserve the assumptions embodied in the design. *Integration* is especially important because individually bounded elements can create new propagation paths when combined. *Verification* asks whether the containment requirements have been satisfied with objective evidence. *Transition* ensures that installation, commissioning, trusted setup, credentials, physical conditions, and the fielded environment preserve the assumptions used during analysis.

Validation asks whether the resulting capability is the right trustworthy secure system for the real mission, including representative compromise scenarios. *Operation* monitors invariants and executes the protective state model. *Maintenance* must prevent temporary tools, vendor access, replacement components, and emergency privileges from becoming a parallel attack architecture. *Disposal* ensures residual credentials, sensitive state, hardware, records, and trust relationships do not create compromise paths subsequent to operational use.

The governing rule is that compromise tolerance is a system life cycle property, not an incident response capability. A security-relevant change should reenter the engineering processes at the level required by its effect on losses, assumptions, containment boundaries, and evidence.

14. A Holistic Systems View of Engineering for Compromise

Engineering for Compromise must extend beyond individual design mechanisms to the broader system context in which trust is allocated, acquired, exercised, sustained, and authorized. Containment can be defeated by decisions and dependencies outside the operational architecture, including acquisition choices, supply chains, human authority, enabling systems, maintenance practices, and shared services. A holistic EfC analysis therefore treats these elements as part of the *security argument* whenever their failure or malicious use can invalidate a system-level claim.

Acquisition, Human Authority, and Trust Allocation

Acquisition establishes trust dependencies long before a system enters operation. Requirements that merely call for trustworthy secure components or compliance with a control baseline may leave the integrator without a precise statement of which *failures* the mission must tolerate. Acquisition documents should instead identify unacceptable losses, required containment properties, independence assumptions, critical trust dependencies, and the evidence expected to show that compromise of selected elements remains bounded. This makes trust allocation an explicit architectural decision rather than an unintended consequence of vendor selection.

Architecture and make-or-buy reviews should examine whether components share administrative roots, management services, update paths, credentials, or other dependencies that could defeat apparently independent barriers. A commercial component may be entirely appropriate inside a containment boundary when the system-level claim does not depend on that component remaining trustworthy. Conversely, an inexpensive shared service can become costly from an assurance perspective when its compromise invalidates

several barriers at once. EfC helps concentrate high-assurance engineering on the small set of elements whose behavior can determine mission loss.

Trust allocation must include people as well as technology. Administrators, operators, maintainers, developers, suppliers, incident responders, and authorizing officials possess credentials, physical access, procedural authority, exception authority, and knowledge of how boundaries are actually operated. The hostile-element question therefore applies to a person or role: if this role were malicious, coerced, deceived, or its credentials stolen, what could it cause? Separation of duties, time-bounded privilege, dual authorization, independent observation, controlled procedures, and immutable evidence become architectural mechanisms when they prevent one human compromise from being sufficient for a consequential action.

Emergency access deserves particular scrutiny. Break-glass mechanisms are intentionally designed to bypass normal restrictions during crises; unless their authority, duration, activation conditions, observation, and termination are explicitly bounded, they become preinstalled compromise paths. A system cannot credibly claim that cross-domain authority is prevented during normal operation while preserving an undocumented emergency procedure that recreates it.

Enabling Systems, Supply Chains, and Maintenance

The fielded system is only part of the trust story. Development environments, compilers, build and signing services, manufacturing equipment, test tools, provisioning services, update repositories, logistics systems, and recovery infrastructure can alter the system before or during operation. If compromise of an enabling system can introduce capability that later bypasses containment, that system belongs within the EfC *security argument* even when it is absent during mission execution.

EfC does not require every supplier and tool to be perfectly trustworthy. It requires engineers to identify which upstream compromises can survive into the operational system and what independent mechanisms can detect or bound their effects. Reproducible builds, provenance records, independent inspection, signed artifacts, diverse verification, constrained update paths, hardware identity, acceptance testing, and configuration attestation can reduce dependence on upstream trust. Claimed diversity must also be tested for common mode: different vendors may rely on the same chipset, library, cloud service, certificate authority, build tool, or subcontractor. Independence depends on the compromise paths that matter to the loss-control claim, not on different product names.

Maintenance is a particularly demanding adversarial condition because systems are deliberately opened, changed, diagnosed, and granted exceptional access. Temporary connections, removable media, vendor laptops, replacement components, debug interfaces, elevated credentials, and physical access can recreate capabilities removed from the operational architecture. EfC therefore treats maintenance as a designed system mode with defined boundaries, authorized scope, time limits, monitoring, evidence requirements, restoration tests, and exit criteria. Temporary capability should be removed or disabled after use, configuration reconciled to an authorized baseline, and critical replacements subjected to renewed identity binding, provenance review, regression verification, and partial reauthorization, as appropriate.

This approach avoids a false choice between unrestricted change and permanent immutability. Eliminating update mechanisms may reduce attack surface, but an inability to repair a discovered vulnerability can force continued operation with a known weakness. EfC favors pre-engineered and constrained change paths that permit necessary evolution while preserving containment, traceability, and the evidence on which authorization depends.

Relationship to Cyber Resilience, Zero Trust, and Traditional Cybersecurity

EfC is closely related to cyber resilience [4] but focuses that discipline on a specific architectural question: **what adversary capability can the system absorb without allowing unacceptable loss**? The cyber resilience goal of *anticipation* identifies compromise conditions and dependencies; the goal of *withstanding* preserves critical

properties while selected elements are untrusted; the goal of *recovery* restores the conditions required for justified trust; and the goal of *adaptation* changes the system without silently invalidating the containment argument. Resilience is therefore evaluated against stakeholder-defined consequences, not merely uptime. A reduced-function protective state may be more resilient than continued availability that permits disclosure, unsafe control, or loss of mission authority.

Zero Trust, defense in depth, secure-by-design development, and conventional cybersecurity remain essential and complementary. Zero Trust challenges implicit trust and emphasizes explicit verification, least privilege, and continuous evaluation. EfC asks what happens after verification fails, credentials are stolen, a trusted service is subverted, or an authorized component becomes malicious. Access decisions alone cannot capture every cyber-physical or mission consequence because an authorized component may possess legitimate capability that becomes dangerous under hostile control.

Traditional cybersecurity reduces the probability, frequency, and duration of compromise; **EfC controls architectural consequence after prevention fails**. Together they establish a continuous loss-control strategy: (1) prevent compromise where practicable; (2) minimize the capability inherited from a compromised element; (3) constrain propagation; (4) detect degradation of containment assumptions; (5) enter protective states before loss thresholds are crossed; and (6) recover an authenticated trustworthy basis. EfC is neither a replacement control framework nor a rebranding of assume breach. It is a loss-driven systems engineering discipline for determining which compromises the mission can tolerate and why.

Measures, Authorization, Limits, and Tailoring

Engineering review and authorization require *measures* that expose the security argument without reducing it to a single score. Useful indicators include the types and combinations of independent compromises required to reach a loss; the size and authority of the trusted footprint; the number and character of cross-boundary interfaces; the proportion of consequential actions requiring distributed authority; the time required to detect loss of a critical invariant and enter a protective state; and the extent to which system recovery depends on infrastructure shared with the compromised domain.

These measures require *engineering judgment*. One poorly understood root of trust may be more consequential than hundreds of compromised endpoints. Additional barriers add little if they share a common dependency, and faster recovery is harmful if it restores compromised state. Authorization should therefore rest on explicit claims: which compromises the system is designed to tolerate, which system elements remain critical trust dependencies, which combinations exceed the compromise margin, what protective actions occur as that margin is approached, and what evidence supports each conclusion.

EfC does not require every system to tolerate arbitrary compromise, nor should compromise margin become a universal compliance number. Physics, cost, performance, timing, and mission constraints may make some elements irreducible trust dependencies. The obligation is to identify those dependencies, apply assurance commensurate with their consequences, and avoid claiming architectural tolerance where it does not exist. *Tailoring* determines the required margin, rigor, evidence, and protective states for each mission and adversary environment.

The resulting argument should be understandable to mission owners: which losses are being prevented, which compromises are expected to remain local, which elements still require exceptional trust, what occurs when confidence declines, and what evidence supports those conclusions. If those relationships cannot be stated clearly, the system may possess many security mechanisms without possessing a coherent *compromise-containment strategy*.

15. Examples of Engineering for Compromise

The principles of EfC are technology-neutral, but their value becomes clearest when applied to representative system classes. The following examples illustrate how the same loss-driven questions (i.e., what becomes reachable after compromise, where propagation is stopped, and which system properties must survive) can be tailored to industrial control systems and critical infrastructure, cloud and enterprise platforms, and national-security and mission-critical systems.

Industrial Control and Critical Infrastructure

Consider an industrial control environment in which a workstation used by an operator is compromised. A prevention-centric analysis concentrates on how malware reached the workstation and how to remove it. EfC begins one step later: assume the workstation is hostile. Can it directly issue commands capable of placing the physical process into an unsafe state? Can it alter controller logic, suppress alarms, change safety thresholds, or impersonate an engineering station? Can it reach the safety system through a shared management service? The answers reveal whether workstation compromise is merely a local cyber incident or a credible path to physical loss.

An EfC architecture would seek to ensure that the operator workstation does not possess unilateral authority for the most consequential physical actions. Safety-critical limits may be enforced by independent controllers. Engineering changes may require a separately authenticated maintenance process/path and independent approval. Cross-zone communications may be constrained to defined commands and states. Protective logic may force the process into a bounded condition when telemetry becomes inconsistent or command sequences violate expected invariants. The essential consideration is *not* the specific control technology; it is the separation between compromise of a general-purpose element and authority to create catastrophic physical consequence.

The same analysis exposes hidden dependencies. If both the production control system and the independent safety system receive configuration from the same engineering laptop, use the same credentials, or depend on the same network time and identity infrastructure, the apparent separation may be illusory. Compromise propagation analysis therefore examines cyber and physical coupling together and treats common maintenance and support mechanisms as part of the containment architecture.

Cloud and Enterprise Platforms

In a cloud or enterprise platform, EfC can begin by assuming compromise of an application workload, service account, administrator credential, or management-plane component. The analysis asks what other tenants, data stores, identities, secrets, deployment pipelines, and administrative functions become reachable. Microsegmentation and identity policy may contribute to containment, but the system claim depends on the complete authority graph, including control-plane APIs, automation credentials, logging systems, backup services, and human operators.

Least privilege is particularly important because modern automation often accumulates broad permissions for convenience. A deployment service that can modify every production environment becomes a high-consequence trust dependency. EfC may lead to partitioned deployment authorities, workload-specific credentials, immutable infrastructure patterns, independent approval for consequential changes, and separate recovery credentials. The architecture should make compromise of one automation function or one workload insufficient to rewrite the entire environment.

Cloud resilience also illustrates why recovery must restore trust rather than merely capacity. Rapidly redeploying a compromised image from an untrusted pipeline reproduces the compromise at scale. Restoring data from backups controlled by the same compromised administrative domain may reintroduce

malicious state. EfC therefore extends containment to build, deployment, backup, and recovery systems and asks whether they share the same compromise boundary as the workloads they are intended to restore.

National-Security and Mission-Critical Systems

For national security and other high-consequence mission systems, the assumed adversary may possess the resources to combine cyber, physical, intelligence, insider, and supply-chain operations over long periods. Under those conditions, the proposition that every defensive layer will remain uncompromised is especially difficult to sustain. EfC provides a disciplined alternative: **identify the mission functions and losses that demand survival, then design so that adversary success against selected elements does not provide sufficient authority to reach those losses**.

This may justify stronger domain separation, physically constrained interfaces, distributed privilege, minimal trusted mechanisms, independent monitoring, and protective states that sacrifice availability to preserve confidentiality, integrity, safety, or command authority. The level of rigor should remain commensurate with consequence. Not every system needs extreme isolation or formal verification, but every consequential system should know which compromises it can tolerate and which ones invalidate its mission security claim.

The Secure Inference Data Center described by RAND provides a useful, specialized example. RAND deliberately assumes that commodity elements and subsystems may be compromised and relies on realm separation, constrained information flows, minimal cross-boundary mechanisms, and layered assurance to keep compromise from automatically producing disclosure or manipulation of protected AI computation [3]. The example is important not because EfC is an AI concept, but because it demonstrates the broader systems principle in a demanding adversary environment.

16. A Repeatable Engineering-for-Compromise Methodology

The preceding concepts can be translated into a repeatable engineering methodology for designing and sustaining systems that remain trustworthy secure under compromise. The following ten steps provide a practical, loss-driven sequence for defining what must be protected, analyzing the consequences of assumed compromise, establishing and testing containment, and maintaining the resulting security claims as the system and its environment change.

- **Step 1 — Define the mission and unacceptable losses.** Determine what the system exists to accomplish, what stakeholders depend upon, and which consequences cannot be tolerated. Establish the claim boundary, adversary assumptions, operating conditions, and accepted tradeoffs.
- **Step 2 — Identify compromise-reachable hazardous states.** Determine which system states, control actions, or combinations of conditions can produce the losses and how adversary control of elements could contribute.
- **Step 3 — Assume selected elements are fully hostile.** Analyze worst-case behavior within the authority, interfaces, information, and physical capabilities available to the element rather than limiting analysis to known vulnerabilities.
- **Step 4 — Determine compromise reach.** Map the privileges, identities, data, functions, interfaces, dependencies, and other domains reachable from each assumed compromise.
- **Step 5 — Establish compromise containment boundaries.** Partition trust, authority, information, control, failure, administration, and recovery so that local compromise does not automatically propagate.

- **Step 6 — Minimize capability within each boundary.** Apply least privilege, least sharing, least functionality, and minimal trusted elements to reduce what the adversary inherits.
- **Step 7 — Mediate necessary interactions.** Make every consequential crossing explicit, bounded, authenticated, observable, failure-aware, and non-bypassable.
- **Step 8 — Analyze propagation and compromise margin.** Determine which single, multiple, correlated, and common-mode compromises can approach or cross unacceptable-loss thresholds.
- **Step 9 — Engineer detection, protective failure, and recovery.** Specify observable invariants, protective states, response authority, evidence preservation, and the conditions required to restore trustworthiness.
- **Step 10 — Substantiate and continuously maintain the claim.** Generate evidence commensurate with consequence and revisit the claim when the mission, configuration, environment, dependencies, adversary, or assumptions change.

17. Conclusion

Cybersecurity should continue to prevent compromise aggressively. Prevention reduces adversary opportunity, raises cost, and remains essential to every credible protection strategy. But for systems whose failure can produce severe mission, economic, safety, or national security consequences, prevention cannot be the final line of reasoning. The architecture must answer what happens when some prevention inevitably fails.

Engineering for Compromise changes the objective from universal component trustworthiness to controlled system consequence. It assumes selected elements can become hostile and then constrains the authority, information, functionality, connectivity, and influence they can acquire. *Domain Separation* establishes containment. *Least Privilege*, *Least Sharing*, and *Least Functionality* limit post-compromise capability. *Mediated Access* and *Distributed Privilege* constrain propagation. *Protective Failure* prevents continued operation from crossing loss thresholds. *Anomaly Detection* observes degradation of the security argument. *Protective Recovery* restores an authenticated trustworthy basis. *Compositional Trustworthiness* provides the evidence that these mechanisms work together as claimed.

The essential distinction is therefore:

Component Compromise ≠ System Compromise ≠ Unacceptable Loss

Architecture determines whether those events become causally connected. A well-engineered system deliberately inserts independent barriers between them and maintains those barriers throughout the system life cycle.

Engineering for Compromise is not an admission that defense has failed. It is recognition that trustworthy secure systems must be designed for the real conditions under which they operate. These conditions include uncertainty, component failure, human error, malicious action, changing dependencies, and determined adversaries. The strongest system is not necessarily the one that claims nothing can be compromised. It is the one that can lose trust in selected elements without losing control of the mission.

A trustworthy secure system is not necessarily one in which nothing can be compromised. It is one engineered so that compromise does not automatically become catastrophe.

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 Trustworthy Secure AI Infrastructure: How NIST SP 800-160 Design Principles and Life Cycle Processes Make RAND's Secure Inference Data Center Trustworthy and Resilient," RONROSSECURE, LLC, 2026.

[3] S. F. Comer et al., Secure Inference Data Centers: A Vertically Integrated Strategy for Security Engineering, RAND Corporation, RR-A4827-1, 2026. https://www.rand.org/pubs/research_reports/RRA4827-1.html

[4] R. Ross et al., *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