

Least Persistence

Limiting Availability and Use to the Time Required for the Intended Purpose

Dr. Ron Ross

CEO, RONROSSECURE, LLC
ISTS FELLOW, DARTMOUTH COLLEGE

Abstract

Least persistence is a foundational security design principle for developing trustworthy secure systems. NIST Special Publication 800-160, Volume 1, Revision 1 states that system elements and other resources are available, accessible, and able to fulfill their design intent only for the time for which they are needed. The engineering implications extend beyond short-lived sessions or routine restarts. Required functions, connections, data, credentials, and supporting resources can remain exposed long after their purpose has ended. This article examines least persistence as a systems security engineering principle. It connects justified duration to stakeholder needs, loss analysis, architecture, system life cycle processes, engineering for compromise, complementary design principles, failure patterns, and assurance evidence. The central proposition is that presence, accessibility, and ability to function must be bounded by need and enforced in the realized system. Removal, disablement, safe modes, and dependable mediation offer different means of establishing those bounds. Trusted representations and trustworthy restoration mechanisms are necessary when resources are removed and reinstated. The objective is sufficient mission capability when required, without unjustified temporal exposure. Least persistence therefore reduces opportunities for misuse and limits the duration of selected adversary capabilities while preserving required continuity, accountability, safety, security, and recovery.

1. The Engineering Problem

Modern systems¹ routinely retain resources beyond the work that justified them. A maintenance connection remains open after the repair is complete. A diagnostic service continues running after commissioning. Temporary credentials become permanent operating dependencies. A processing job leaves sensitive information in memory, storage, logs, and backups. An AI agent retains access to tools after its assigned task has ended. Each resource may have been legitimate when introduced. The engineering problem is determining when that legitimacy ends and ensuring that the resource no longer remains available for unjustified use.

Time changes the security significance of an otherwise acceptable architecture. A narrowly scoped maintenance interface can still provide recurring opportunities for intrusion if it is continuously accessible. A correctly authorized session can become an avenue for misuse when it outlives its purpose. A protected copy of information can become unnecessary exposure when retained indefinitely. Functional need establishes why a resource exists; temporal need establishes when it should exist, be reachable, or remain able to act.

“A capability justified for one task should not become a standing opportunity for the next adversary.”

¹ Systems include information technology (IT) systems, operational technology (OT) systems, Internet-of-Things (IoT) devices, and cyber-physical systems.

Convenient implementation choices often obscure these distinctions. Engineers leave services active to avoid startup delay, preserve broad connectivity for troubleshooting, retain temporary information for possible future use, and configure automatic renewal to avoid interruptions. Those implementation choices can be reasonable, but their consequences require explicit analysis. Convenience should not determine the duration of exposure for a mission-critical resource.

The problem extends beyond a visible process or account. A terminated process may leave reusable credentials. A disconnected session may leave a queued command that still executes. A refreshed workload may reconnect to corrupted persistent storage. A disabled hardware function may be reactivated through a management interface. System analysis must address what remains present, accessible, and usable after the nominal end of the task, including indirect paths through other system elements.

Cyber-physical systems make the trade particularly consequential. Removing power from a controller may eliminate one opportunity for misuse while also removing the ability to maintain a safe physical state. Repeatedly restarting a service may disrupt required continuity or erase information needed for protective action. Least persistence must therefore be engineered around mission and protection needs, rather than imposed as a blanket instruction to disconnect, delete, or restart everything.

Frontier AI adds a practical concern without changing the principle. A compromised agent can use tools, connections, and retained information within the time they remain available. Ending a task in a user interface does not necessarily end the associated machine authority or background work. The architecture should define the completion event, establish an enforceable end to the relevant capabilities, and account for information and effects already produced.

The objective is to make justified duration an architectural property. Engineers must establish when resources are required, what conditions end that need, which mechanisms enforce the boundary, and how required capability is restored in time. A timeout setting is one implementation detail within that larger argument. The system must demonstrate that the intended restriction survives errors, transitions, and the compromises it is designed to tolerate.

2. What the Principle Means

NIST SP 800-160, Volume 1, Revision 1 states the design principle of *Least Persistence* as follows [1]:

“System elements and other resources are available, accessible, and able to fulfill their design intent only for the time for which they are needed.”

NIST explains that least persistence reduces susceptibility by limiting how long functions, resources, data, and information remain present, accessible, and usable when not required. This reduces opportunities for inadvertent or unauthorized use, modification, or activation. The scope includes active elements that execute and non-active resources, such as stored information, that can be accessed or acted upon without themselves executing.

Three conditions organize the analysis. Presence concerns whether a system element or resource is installed, loaded, resident, and configured. Accessibility concerns whether it can be invoked, interacted with, or operated on. Ability to function concerns whether an active system element is powered, enabled, or armed to execute. NIST applies presence and accessibility to active and non-active resources, and ability to function to active system elements. The engineering record should identify which condition is being constrained and by what mechanism.

- **Presence.** Instantiate software only when required; remove temporary processing artifacts when their purpose ends; avoid retaining unnecessary operational copies of sensitive information. Analyze where copies reside and which dependencies require continued retention.
- **Accessibility.** Establish a maintenance connection for an authorized activity and terminate access when that activity ends. A retained system element can remain inaccessible outside its justified interval if dependable mediation enforces the restriction.
- **Ability to function.** Remove power, disable operation, or establish an appropriate safe mode when execution is unnecessary. The resulting state must preserve required mission and protection behavior, including any physical safety obligations.

NIST establishes a practical progression. The broadest interpretation avoids installing, instantiating, or powering system elements and resources until needed, then removes them or their power when no longer required. Where this is not possible or practical, the elements or resources should be disabled, disarmed, or placed in a safe mode that prevents their functioning or use. Where those specific measures are not feasible, *Mediated Access* constrains the time and duration of use [1]. These alternatives support different claims and should not be described interchangeably.

“Least persistence makes the end of need an enforceable boundary.”

Need can be event-driven as well as time-driven. An authorized maintenance task may end on completion, cancellation, loss of supervision, transition to operational mode, or expiration of a maximum interval. An information resource may remain necessary until a transaction is completed, a recovery obligation ends, or an accountability requirement is satisfied. Calendar time alone does not capture these conditions. The requirements should specify both the relevant events and any maximum permitted duration.

A duration limit requires an explicit interpretation. Does it constrain access to an interface, acceptance of a request, continued execution, possession of a credential, or retention of the resulting information? Expiring a credential after request acceptance may not stop work already underway. Terminating a connection may not cancel a command already delivered. Each claim must identify the activity being bounded and the point at which the restriction takes effect.

NIST also identifies benefits for cleansing corrupted aspects, restoring a known state, and minimizing exposure to the environment, attack, and erroneous behavior. These benefits depend on implementation. When system elements are removed and restored, NIST requires a trusted representation and a trusted ability to instantiate them within the time constraints for use [1]. A short lifetime without a trustworthy restoration basis can repeatedly reintroduce the same corruption.

The security design principle of *Least Persistence* complements the design principles of *Least Functionality* and *Least Privilege*. *Least Functionality* concerns capability required for purpose; *Least Privilege* concerns authority allocated to accomplish specified functions; *Least Persistence* concerns justified availability and use over time. A required capability with narrow authority may still be unnecessarily available between tasks. Conversely, a short-lived element can still possess excessive functionality and privilege during its entire lifetime. Effective design addresses all three dimensions.

3. What the Principle Does Not Mean

Least persistence is often associated with temporary workloads, expiring credentials, and periodic refresh. Those mechanisms can support the principle, but their names do not establish that the required temporal restrictions actually hold.

- **Least persistence does not mean the shortest possible lifetime regardless of purpose.** Mission execution, safety, protection, accountability, and recovery establish legitimate needs. Removing a required resource before those needs end can cause the unacceptable loss the architecture is intended to prevent.
- **Least persistence does not mean every resource must disappear physically.** NIST permits disablement, safe modes, and mediated restrictions where removal is impractical. The claim should distinguish absence from retained but inaccessible or unusable resources, including any reactivation paths.
- **Least persistence does not mean periodic restart establishes trustworthy recovery.** A restarted service may load the same compromised image, configuration, secrets, or persistent data. Trustworthy restoration requires evidence about the source, the restoration mechanism, and the state admitted into operation.
- **Least persistence does not mean an expired session reverses its effects.** Commands already accepted, information already disclosed, and changes already committed can survive the session. The design must separately bound consequential operations and their resulting state.
- **Least persistence does not mean information can be deleted without regard to obligations.** Required records, evidence, safety history, and recovery data may need retention. Their duration, location, accessibility, and protection should be justified rather than treated as an exception with no boundary.
- **Least persistence does not mean a temporary identity is harmless.** During its valid interval, an identity can exercise every privilege it possesses. A renewable credential can provide effectively continuous access if renewal is unrestricted or controlled by the compromised holder.
- **Least persistence does not mean a timer controlled by a compromised process independently constrains that process.** A dependable external mediator, protected platform mechanism, or appropriate hardware restriction may be required under the stated compromise model.

The term persistence also needs care. *Least Persistence* is a design principle about the duration of legitimate system elements and resources. Adversary persistence concerns maintaining a foothold or continued *influence*. Applying the principle can disrupt selected footholds and reduce their opportunities, but it does not establish that all adversary persistence mechanisms have been eliminated. An adversary can move into retained state or a supporting element before a temporary workload is removed.

Similarly, a resource described as ephemeral may depend on continuously available management, storage, identity, and build services. The temporary resource's lifetime says little about the exposure of those dependencies. The architecture must include the system elements that create, authorize, renew, terminate, and restore it. Otherwise, the apparent temporal restriction can be bypassed through the infrastructure responsible for enforcing it.

Function termination and safe termination are different engineering obligations. Abruptly stopping a physical actuator, protective monitor, or control service can be hazardous. The design should identify the point at which new actions are prohibited, how ongoing actions conclude or transfer safely, and which protection functions remain active. Least persistence supports justified duration; it does not waive the need to manage transitions correctly.

4. Loss-Driven Interpretation

Least persistence begins with the losses stakeholders consider unacceptable. NIST's loss-driven systems security engineering approach establishes the system purpose, assets, protection needs, and conditions that can produce unacceptable loss [1]. Those considerations determine which resources must remain available and which continuing exposure is unjustified. The duration of a resource should follow from that analysis rather than from a vendor default or a universal timeout policy.

Consider a water treatment facility. Remote maintenance may be needed to diagnose a controller during an approved work interval. A continuously available maintenance connection can expose the controller to unauthorized changes outside that interval. But ending access during an incomplete repair can also leave equipment in an unsafe condition. The engineering task is to define acceptable maintenance states, permitted actions, safe completion or abort behavior, and the independent mechanisms that preserve those conditions.

Different resources have different end conditions. A temporary command channel may close immediately after an authorized transaction. Sampled operational data, such as a water-quality reading, may need to remain accessible until its accuracy has been checked and discrepancies with related readings or records have been resolved. An audit record may require much longer retention in a separately protected repository. A safety controller may need continuous operation throughout the mission. The principle does not require identical lifetimes; it requires justified ones.

The analysis should distinguish the duration of *exposure* from the duration of *consequences*. A brief write opportunity can produce a durable configuration change. A temporary disclosure can reveal information that cannot be recalled. A short period of actuator control can cause a physical loss before any timeout occurs. Least persistence reduces temporal opportunity, but other principles must constrain what can happen while that opportunity exists.

- **Identify the loss pathways.** Determine how continued presence, accessibility, or operation could enable disclosure, corruption, unsafe activation, interference, or unauthorized influence. Include residual state and indirect use after the visible activity ends.
- **Establish the required interval.** Specify the earliest necessary availability, the event that ends need, and the latest acceptable completion of restriction. Include startup, supervision, shutdown, and restoration time where those affect mission or safety.
- **Determine the tolerable effects.** Bound actions and state changes during the interval. A duration limit is inadequate when the permitted action can immediately cause unacceptable loss.
- **Identify dependent obligations.** Preserve necessary accountability, evidence, recovery, and physical protection. Allocate those obligations to appropriate resources with their own justified duration and access conditions.
- **State the assumptions.** Identify the clock, mediator, restoration source, management authority, and platform dependencies on which the temporal claim relies. Explain which failures or compromises invalidate the claim.

A useful engineering question is whether the system remains acceptably protected if the resource becomes hostile at the beginning of its permitted interval. If the adversary can cause unacceptable loss before the interval ends, shortening the interval alone will not solve the problem. The architecture needs restrictions on capability, authority, propagation, and effects, as well as time. This is where least persistence connects directly to *Engineering for Compromise* [2].

“A shorter opportunity for attack is valuable only within an architecture that also bounds what the attacker can do.”

Time limits should be selected through analysis and validated under realistic conditions. Average task duration is insufficient if startup delays, lost communications, clock faults, or protective transitions can extend exposure beyond the accepted bound. Conversely, a limit that routinely interrupts legitimate work encourages bypasses and permanent exceptions. Requirements should reflect both the worst credible operational conditions and the consequences of failed enforcement.

Loss-driven interpretation also prevents retention policy from becoming indiscriminate deletion. The question is which information remains necessary, for what purpose, in which form, and accessible to whom. Moving a required record into protected archival storage can reduce operational exposure while retaining its legitimate value. The record remains persistent, but its operational accessibility and use are constrained according to need.

5. Engineering the Principle into the Architecture

Least persistence must be allocated to mechanisms and relationships the system can enforce. An architectural description should show the relevant resources, their permitted intervals and modes, the conditions for activation, the means of termination, the treatment of residual state, and the trusted basis for restoration. It should also show who can change those conditions. A diagram of temporary workloads without their management and storage dependencies is incomplete.

- **Define temporal contracts for resources.** Record the purpose, activation authority, start conditions, end conditions, maximum duration where applicable, permitted extensions, and required terminal state. Distinguish the lifetime of a system element from the lifetime of its access, information, effects, and authority.
- **Separate enforcement from the restricted system element.** Where the compromise model requires independent termination or denial of renewal, allocate those functions to protected mechanisms outside the restricted element's control. Analyze the restricted element's ability to change clocks, policies, identities, queues, or termination commands.
- **Mediate activation and renewal.** Require a justified request to instantiate, connect, enable, or renew a resource. Treat extension as a new consequential decision when it prolongs exposure. Unrestricted self-renewal can defeat an otherwise narrow lifetime.
- **Engineer orderly termination.** Stop acceptance of new work, account for in-flight operations, preserve required evidence, and transfer or conclude physical control safely. Establish a bounded result when normal shutdown is blocked or incomplete.
- **Control residual information and authority.** Address temporary files, memory, caches, credentials, snapshots, diagnostic records, and shared storage. Determine which must be removed, invalidated, retained under separate protection, or admitted through validation into future operation.
- **Protect restoration and administration.** Trusted images, configuration stores, deployment controllers, identity services, policy stores, and update mechanisms determine what returns and for how long. Their compromise can invalidate restrictions on an otherwise correctly designed temporary element.

The strongest feasible mechanism depends on the resource and its purpose. Removing a processing instance can end its execution, but retained credentials and accessible data can still permit influence through a different instance. Removing power from a device can end active function while leaving stored information readable through other means. Restricting accessibility can preserve a needed resource while preventing its ordinary use outside a justified interval. Each implementation needs evidence for the specific condition it claims to establish.

Time enforcement requires dependable semantics. A maximum session duration differs from an inactivity timeout. Activity generated by an adversary may continually reset the latter. A calendar expiration differs from a duration measured from activation. Clock adjustment, synchronization loss, suspend and resume, and reboot can affect these mechanisms differently. The architecture should identify its time source and define how uncertainty or loss of that source affects continued availability—including explicit protections against an adversary deliberately manipulating or desynchronizing system clocks to artificially extend temporal grants.

Revocation must reach the place where authority is exercised. A central record may say that access has expired while an endpoint accepts a cached authorization or a previously issued credential. A disconnected device may continue applying old permissions. The system should specify the maximum delay between termination of authorization and enforcement at the resource, including degraded conditions. The claim is about effective access, not merely the state of an administrative database.

An illustrative maintenance architecture uses a protected access service to approve a bounded connection to a selected controller. The controller independently permits only the maintenance operations justified for its current mode. A separate termination mechanism closes access at completion or expiration, and the controller validates changes before returning to operation. Required records are transferred to protected storage. This arrangement combines temporal restrictions with specific limits on capability, privilege, and admissible state; the approved interval alone does not provide those other protections.

For AI-assisted operations, the same reasoning can apply to a task environment. Tool access is established for an approved task, narrowly allocated, and terminated by a mechanism the agent cannot extend. Background jobs, delegated identities, and generated credentials are accounted for. System architectures must also account for information retained in agent context, memory stores, caches, and supporting services. Sensitive query outputs, API tokens, and intermediate task data should be removed or made inaccessible when their justified use ends, with any continuing retention separately justified and protected. Outputs requiring operational use pass through appropriate approval or validation. Selected records remain available for accountability under a separate retention policy. Ending the agent process is only one part of ending its effective influence.

Restore capability within the mission's timing constraints. On-demand provisioning that cannot meet required availability is not an acceptable implementation. Prepositioned protected images, staged standby resources, or retained system elements with mediated access may be necessary. The trade should be explicit: retain what continuity requires, constrain its unnecessary exposure, and demonstrate that activation establishes an acceptable state in time.

“The lifetime of a process does not bound the lifetime of its credentials, data, or effects.”

6. Engineering for Compromise

Engineering for Compromise assumes selected elements may become malicious, lose integrity, or fall under adversary control and then asks whether unacceptable loss remains prevented [2]. The design principle of *Least Persistence* contributes by limiting how long selected resources and capabilities are available to the compromised system element. It can also simplify removal and reestablishment of operational elements. Those benefits require an explicit account of what the adversary can control and what remains trustworthy.

Assume a temporary processing service is compromised immediately after launch. A protected supervisor may terminate the service at its allotted end time. That termination can constrain further execution in the service, but the adversary may already have modified shared storage, copied credentials, submitted work elsewhere, or changed a future deployment. The compromise analysis should trace each path from temporary control to continued influence. Removing the original process does not automatically remove its consequences.

The *Engineering for Compromise* paper's Compromise Propagation Analysis provides a useful analytical approach [2]. Begin with the system element already hostile and ask what becomes reachable through identity, information flow, control flow, configuration, software distribution, administration, and recovery. For least persistence, add the temporal questions: what remains reachable after expiration, which actions can be scheduled for later, and what state will a replacement element inherit? The analysis should expose durable paths created during a short-lived compromise.

The compromise model determines the *value* of the restriction. A timer inside a hostile workload may be ignored. A supervisor can remain effective under workload compromise but fail under compromise of the host platform. A physically separate access mediator may enforce the permitted duration of a connection, but its effectiveness still depends on protected configuration and a trustworthy timing mechanism. The security argument should identify those distinctions rather than claim that a timeout survives every form of compromise.

Renewal deserves particular attention. If a compromised system element can obtain replacement credentials, launch successor instances, or ask a shared controller to extend its task without independent authorization, the architecture may provide continuous capability through a sequence of temporary grants. Individual lifetimes remain short while effective adversary access remains *persistent*. Enforcement must address succession and delegation as well as the original resource.

Refreshing a service can remove selected corrupted execution state when the platform and restoration basis remain trustworthy and hostile state is not reintroduced. However, an adversary may compromise the image repository, boot source, configuration, dependency, or retained data. A refresh can then reliably recreate the adversary foothold. *Engineering for Compromise* treats build, deployment, backup, and recovery systems as consequential dependencies with their own separation, authority limits, and evidence [2]. Least persistence increases the importance of these dependencies when restoration becomes frequent.

“A fresh instance is trustworthy only if its source, restoration path, and admitted state justify that confidence.”

Termination should also be analyzed as an adversary-triggerable event. A hostile system element may provoke repeated expiration, consume restoration resources, interfere with cleanup, or create inconsistent state during shutdown. The system needs robust protection against these effects where they can lead to unacceptable loss. An architecture that contains corruption but lets an adversary continuously deny an essential service has not necessarily achieved its *mission protection* objective.

For cyber-physical functions, trusted safety and protective control may need to remain active while compromised operational capability is removed. The design should identify which physical state must be maintained, which authority is withdrawn, and which independent mechanism preserves the invariant. Recovery should verify the relevant logical and physical conditions before returning control. Restarting software cannot by itself establish that the physical process is ready for resumed operation.

Least persistence therefore supports containment without replacing it. It reduces opportunities for selected misuse and can bound selected continuing actions, but an adversary may act faster than the permitted interval. The design principles of *Domain Separation*, *Least Functionality*, *Least Privilege*, and *Mediated Access* restrict the actions that can occur—and the losses they can cause—during that interval. The principles of *Protective Failure* and *Protective Recovery* preserve required protection if failures occur and as operation is restored. The engineering claim should state which continuing influence is ended, which remains possible, and why the remaining exposure is acceptable.

7. Interaction with Other Principles

Least persistence gains its value through composition with other security design principles. The first four principle articles establish complementary architectural questions about separation, access, authority, and capability. Least persistence adds the duration of justified availability and use. It should be integrated with those questions rather than treated as an isolated configuration practice [1].

- **Domain Separation.** Separate temporary execution and its state from protected operational, administrative, and recovery resources. Separation helps prevent a short-lived compromise from creating a foothold in a longer-lived domain. Shared dependencies and interfaces remain part of the analysis.
- **Mediated Access.** Enforce the conditions under which a system element or resource is accessible and usable. Temporal mediation should cover activation, continued use, renewal, and termination as required by the policy. Enforcement must occur at the consequential access points.
- **Least Privilege.** Limit what a temporary identity or system element can accomplish while authorized. A short interval with excessive authority can still permit immediate loss. Scope and duration should be justified together, including delegation and successor identities.
- **Least Functionality.** Provide only the capability needed for purpose. Temporarily exposing an unnecessary function does not make it necessary. Conversely, a required function need not remain available at all times or in every mode.
- **Least Sharing.** Reduce the common state and mechanisms through which a removed system element can influence its replacement or other elements. Shared writable storage, identities, and management services can carry corruption across otherwise separate lifetimes.
- **Protective Defaults.** Establish default conditions that prevent unjustified activation, continued availability, or automatic renewal. Restart, reset, and recovery should not restore standing exposure that the operational policy removed.
- **Protective Failure.** Define an acceptable response when expiration, revocation, cleanup, timing, or termination cannot be assured. The protective state must be appropriate for the mission; abruptly removing essential physical control may be unacceptable.
- **Protective Recovery.** Reestablish trustworthy conditions before system capability returns. Trusted representations, protected restoration authority, validated configuration, and controlled admission of retained state support restoration beyond merely restarting execution.
- **Continuous Protection.** Preserve necessary protection throughout activation, normal use, shutdown, retained states, restarts, and restoration. Least persistence can reduce the duration of exposure while continuous protection addresses the intervals and transitions that remain.
- **Compositional Trustworthiness.** Substantiate that local lifetime claims survive integration. A workload's timeout is not a system-level limit if another service can renew it, retained credentials remain valid, or persistent state carries its influence forward.
- **Distributed Privilege.** Protect consequential extensions and restoration decisions by allocating authority so that one compromised element cannot exercise the complete decision alone.
- **Anomaly Detection.** Identify resources that remain available beyond their permitted interval or reappear unexpectedly.
- **Substantiated Trustworthiness.** Provide evidence supporting claims that resources remain available only for their justified duration and that required protection is preserved.

Their application should reflect the loss consequences and the specific architecture rather than a checklist of principle names [1].

NIST SP 800-160, Volume 2, Revision 1 provides a related cyber resiliency technique, *Non-Persistence*, defined as generating and retaining resources as needed or for a limited time [3]. Its approaches are *Non-Persistent Information*, *Non-Persistent Services*, and *Non-Persistent Connectivity*. These provide concrete means of limiting information exposure, refreshing or terminating services, and establishing connections on demand. The technique can support *Least Persistence*, but its implementation still requires the principle's engineering justification.

The distinction matters. Periodic refresh under *Non-Persistent Services* can reduce the time selected corrupted execution state remains active, yet a continuously required service may be replaced without reducing continuous system-level availability. *Non-Persistent Information* can bound temporary operational copies while required archival records persist under protection. *Non-Persistent Connectivity* can terminate a connection without withdrawing credentials or cancelling accepted work. The design should name the relevant approach and state the precise property it supports.

Other cyber resiliency techniques, including *Segmentation* and *Substantiated Integrity*, can help protect the domains and restoration basis on which temporal restrictions depend [3]. Selecting a technique does not substantiate a claim by itself. The architecture, implementation, and evidence must demonstrate that the selected mechanism achieves the needed restriction under the identified operating and compromise conditions.

8. System Life Cycle Application

Least persistence is a life cycle obligation. NIST SP 800-160, Volume 1, Revision 1 provides technical processes for translating stakeholder concerns into requirements, architecture, implementation, evidence, operation, and eventual disposal [1]. Temporal restrictions should remain traceable across those processes. If they appear only in operating procedures after the architecture is fixed, implementation constraints can make dependable enforcement difficult.

- **Business or Mission Analysis.** Identify when critical capabilities are required and which losses matter. Examine the mission activities that create temporary exposure, including commissioning, maintenance, emergency access, incident response, supplier support, and decommissioning.
- **Stakeholder Needs and Requirements Definition.** Establish the continuity, safety, accountability, and recovery obligations that justify persistence. Define stakeholder expectations for required availability and withdrawal, and explain the consequences of both excessive duration and premature withdrawal.
- **System Requirements Definition.** State measurable temporal conditions. Identify activation events, required availability, maximum duration where appropriate, completion and cancellation events, enforcement delays, permitted extension, and resulting states. Define the handling of in-flight work, timing uncertainty, loss of connectivity, restart, and restoration. Avoid requirements that merely say “temporary access” without establishing an observable end condition.
- **Architecture Definition.** Allocate temporal conditions to resources, mediators, protected time sources, supervisors, management authorities, and restoration mechanisms. Identify the relationships and boundaries that support enforceable activation, continued use, renewal, and termination.
- **Design Definition.** Specify how allocated mechanisms implement resource lifetimes, end conditions, safe transitions, and trustworthy restoration. Make assumptions about platform trust and supporting services explicit, particularly where many temporary resources share a common enforcement or restoration mechanism.

- **System Analysis.** Evaluate the trades among exposure, continuity, latency, physical safety, retained information, and enabling-system dependence. Determine whether the proposed restrictions preserve required mission and protection behavior under the relevant operating and compromise conditions.
- **Implementation.** Ensure that timers, completion events, revocation, cleanup, and mode transitions have the specified behavior. Remove unintended renewal and reactivation paths. Protect the configuration and code that establish resource lifetimes.
- **Integration.** Verify temporal properties across identities, services, devices, storage, networks, and management systems. Confirm that the termination in one system element produces the required restriction at dependent elements and does not merely update a local record.
- **Verification.** Demonstrate required availability and bounded withdrawal under the specified conditions. Exercise failures, expiration, extension, queued operations, clock changes, and retained state. Test the mechanisms expected to constrain a compromised system element.
- **Transition.** Establish approved operational baselines, remove commissioning exposure, validate restoration sources, and ensure that temporary supplier or migration access ends. Preserve necessary records under the intended retention and protection arrangements.
- **Validation.** Determine whether the realized restrictions support stakeholder needs in actual mission scenarios. Confirm that operators can complete required work and safely respond when a resource expires or cannot be restored.
- **Operation and Maintenance.** Monitor departures from approved duration and mode conditions. Reassess temporary exceptions, auto-renewal, changed task duration, supplier updates, new persistent dependencies, and emergency access. Maintain evidence for effective enforcement.
- **Disposal.** Withdraw connections and authority, address retained information, and remove operational and recovery paths that are no longer justified. A retired system element should not remain a live dependency or a source of credentials and sensitive state.

The life cycle processes of *Configuration Management* and *Information Management* support these technical activities by maintaining the baselines, approved exceptions, retention decisions, and evidence. The *Risk Management* process informs the treatment of unavoidable exposure. The systems engineering record should make clear whether an exception changes the intended restriction, changes the threat or compromise assumptions, or accepts a specific residual risk. An exception that is repeatedly renewed should receive substantive review rather than become an undocumented permanent condition.

Acquisition and supplier relationships also matter. Determine whether supplied components support bounded sessions, reliable revocation, controlled activation, removal of temporary state, and trustworthy restoration. Establish what can be verified by the acquirer and what depends on supplier *evidence*. A component that cannot disable a maintenance capability may require an external mediator; a device that retains information may require a different operational allocation or disposal approach.

As the mission evolves, justified duration can change. A diagnostic resource once continuously needed may become unnecessary after stabilization. A new safety obligation may require longer retention. A software update may add automatic renewal or change inactivity behavior. Such changes require renewed analysis and appropriate verification. The baseline should describe current need, rather than accumulate the duration assumptions of earlier versions.

9. Failure Patterns and Anti-Patterns

Least persistence failures often arise when a nominal end event is mistaken for an effective end to capability. The following patterns deserve explicit attention in architecture reviews, operational assessments, and assurance activities.

- **The temporary exception becomes permanent.** Maintenance access, diagnostic operation, or emergency authority remains enabled because no one owns its closure. Establish an accountable end event, enforceable duration, and reviewed extension path.
- **The restricted element owns its deadline.** A workload can reset its timer, modify its expiration setting, disable its supervisor, or continually signal activity. Allocate dependable enforcement outside the system element where the compromise assumptions require it.
- **Expiration is recorded but not enforced.** Central authorization ends while cached permissions, local sessions, or issued credentials remain usable. Establish the effective bound at each consequential resource, including degraded and disconnected operation.
- **Short-lived grants create continuous access.** Automatic renewal, successor identities, delegated jobs, or repeated instantiation allow the same effective capability to continue. Analyze the chain of grants and require justified renewal and succession.
- **Shutdown ignores accepted work.** A session ends while queued commands, scheduled actions, or ongoing transactions remain consequential. Define cancellation, completion, safe transfer, and post-termination behavior for work already admitted.
- **The resource disappears but its state survives.** Temporary files, shared volumes, snapshots, secrets, and diagnostic copies preserve data or malicious changes. Trace residual state and assign justified retention, protection, removal, or validation requirements.
- **Refresh reintroduces corruption.** A replacement trusts compromised images, configuration, persistent data, or management authority. Protect and substantiate the restoration basis, including the state brought forward from prior operation.
- **Removal disrupts protection.** Expiration stops a safety monitor, interrupts necessary control, or discards evidence before it is secured. Allocate continuing protection and retained obligations before withdrawing the temporary resource.
- **Timing uncertainty becomes unlimited access.** Clock failure, synchronization loss, reboot, or communications failure causes indefinite continuation. Define a bounded protective response consistent with the mission's continuity and safety needs.
- **Recovery restores standing exposure.** Default configuration reopens access, reenables diagnostics, or regenerates broad credentials. Verify that restoration reestablishes approved temporal conditions before operational use.
- **The inventory stops at the application.** Temporary workloads are counted while identity, storage, supplier support, administration, and recovery dependencies remain unanalyzed. Include the enabling systems that determine effective duration and future influence.

These patterns can combine. A maintenance session may expire correctly, yet the technician's temporary credential may remain valid in another domain. A compromised process may be removed, yet a scheduled task launched through a shared controller may recreate it. A replacement may load approved code while reading attacker-modified configuration. Review should trace the entire capability and state transition rather than infer system-level protection from one successful timeout.

Operational pressure can make these failures recur. Repeated interruptions encourage staff to lengthen intervals, bypass controls, or preserve access between tasks. The appropriate response is to revisit the requirement and design, identify legitimate work patterns, and provide an enforceable solution that supports them. A fragile restriction dependent on continual manual discipline may not provide confidence commensurate with the consequences of failure.

“Temporary by intention is not temporary by enforcement.”

Claims should avoid substituting labels for evidence. “Ephemeral,” “automatically refreshed,” “just in time,” and “expires after use” identify intended behaviors, not their achieved limits. Each should be translated into a specific statement about presence, accessibility, operation, authority, or retained state, accompanied by the conditions and evidence that support it.

10. Evidence and Assurance

A least-persistence claim requires evidence that system elements and other resources remain available, accessible, and able to fulfill their design intent only for the time they are needed. A timeout setting, process termination record, or credential expiration policy demonstrates part of an implementation. It does not by itself establish complete coverage, justified duration, effective withdrawal, safe transitions, or trustworthy restoration.

A top-level claim might be:

Each system element and other resource within the defined scope is available when required for its intended purpose, and its presence, accessibility, and ability to function are limited to justified intervals under the specified operating conditions, life cycle transitions, and failure and compromise assumptions.

Supporting subclaims should address the completeness of the resource and dependency inventory; adequacy of temporal requirements; traceability of justified durations; correctness of activation and termination; enforcement delays; renewal and extension authority; alternate access paths; delegated and queued work; residual credentials and information; operating modes; composition; safe transitions; and trustworthy restoration. Exceptions and continuing retention obligations should be visible in the claim rather than concealed behind an assertion that resources are temporary.

Evidence may include requirements-to-resource traceability, architecture models, resource inventories, state-transition models, activation and termination specifications, timing analyses, session and credential policies, configuration records, retention schedules, dependency analyses, code review, access tests, adversarial exercises, fault-injection results, operational drift records, and recovery demonstrations. High-consequence boundaries may warrant formal analysis of temporal properties or systematic analysis of reachable states and transitions.

Verification should show that every required resource becomes available within its specified timing constraints and remains available for the work that justifies it. Include startup, normal operation, errors, maintenance, degraded conditions, shutdown, and recovery as applicable. A resource that expires correctly during a routine test may still terminate too early during a necessary protective action. Validation should examine whether operators can sustain the mission and preserve safety without bypassing the temporal restrictions.

Verification should also demonstrate that unjustified availability and use end within the specified enforcement delay. Exercise normal completion, cancellation, maximum-duration expiration, loss of supervision, and withdrawal of authorization. Attempt continued use through existing sessions, cached

permissions, retained credentials, alternate interfaces, delegated jobs, and queued commands. Where removal is claimed, examine residual information and dependencies as well as observable behavior. Where disablement or mediation is used, test reactivation and bypass through configuration changes, reset, update, recovery, and authorized mode transitions.

An effective time limit is a demanding claim. A log entry stating that a session expired does not prove that the consequential resource stopped accepting its requests. A process termination event does not establish that its credentials, delegated work, or retained state can no longer be used. Stronger evidence combines implementation and configuration records, analysis of relevant continuation paths, and observations at the interfaces and state locations where the restriction must take effect. The argument should identify timing uncertainty and what is not known, especially for supplier-controlled services or mechanisms whose internal behavior cannot be inspected.

Compromise-oriented verification should match the stated adversary model. For a compromised operational process, attempt to reset its timer, renew its credentials, launch successor work, preserve accessible state, or interfere with termination. For a compromised administrator, attempt changes to duration settings, renewal policy, timing sources, and restoration inputs within that administrator's authority. Determine which restrictions survive and which depend on the compromised system element's cooperation. Tests should not assume that physical separation alone establishes dependable enforcement; the mediator's configuration, timing, management, and supporting dependencies must also support the claim.

An illustrative remote maintenance connection to a water treatment controller can make the argument concrete. Assume maintenance access is required for an approved task, while protective monitoring and necessary physical control must continue throughout the activity and after access ends. The connection is permitted only during the authorized work interval and must terminate within a specified enforcement delay following completion, cancellation, or expiration. Records needed for accountability remain in a separately protected repository under their own retention and access requirements.

The architecture can use a separately protected access mediator to establish and terminate the connection. Temporary credentials expire or are revoked at the resources where they are accepted, and extending the work interval requires renewed authorization. The design specifies whether commands already accepted are completed, canceled, or transferred safely. If access ends before maintenance is complete, the controller must enter or remain in a condition that preserves required protection. Any later reconnection requires an authorized activation path.

Verification for this example includes demonstrating timely access for approved maintenance, observing termination at the controller, attempting use of out-of-date credentials and existing sessions, testing renewal without approval, and examining queued-command behavior. Fault tests address communications loss, timing failure, mediator restart, and interrupted restoration. Compromise testing assumes malicious use of the capabilities available during the authorized interval and attempts to preserve access or influence afterward. Those tests support the wider containment argument; least persistence alone does not establish that an authorized maintenance action cannot produce an unsafe or durable change.

“An expiration setting states an intention. An assurance argument demonstrates when capability ends and what remains afterward.”

An *assurance case* connects claims, reasoning, evidence, assumptions, and limitations. It should identify which enforcement mechanisms remain trusted, which compromises are tolerated, what supplier knowledge is available, and which operating modes change justified availability. The assurance argument

must distinguish the lifetime of a connection or process from the lifetime of its authority, effects, and information. Required records and recovery data may remain under separately justified protection arrangements. Information already disclosed and changes already committed may survive termination. Reducing temporal exposure is an engineering contribution to protection, not automatic proof that unacceptable loss is prevented.

Evidence must be maintained as the system evolves. A new dependency, automatic renewal mechanism, agent tool, supplier interface, retention policy, configuration setting, or recovery image can materially alter the temporal baseline. Such changes require renewed analysis and appropriate verification. Operational observations help detect drift, but the absence of an observed overrun or misuse does not establish that every unexamined continuation path is constrained.

11. Key Engineering Takeaways

1. The principle of *Least Persistence* applies to system elements and other resources, including functions, data, information, connectivity, and supporting capability. Its scope extends beyond the lifetime of a visible software process.
2. Justified duration follows from mission and protection needs. Required accountability, continuity, safety, and recovery must remain achievable; the objective is no unjustified temporal exposure.
3. Presence, accessibility, and ability to function are distinct conditions. Identify which is constrained and distinguish removal from disablement, safe modes, and temporal mediation.
4. Specify activation and end events, maximum duration where appropriate, enforcement delay, extension authority, and terminal state. A resource described only as temporary has no demonstrated temporal boundary.
5. Constrain renewal, delegation, queued actions, and successor resources. Short-lived grants can collectively provide continuous adversary capability if those paths are unrestricted.
6. The lifetime of a system element does not establish the lifetime of its data, authority, or effects. Analyze residual state and the durable consequences of actions taken during the permitted interval.
7. *Engineering for Compromise* tests whether temporal restrictions survive adversary control. The mechanism that terminates or denies continued use must remain trustworthy under the stated compromise assumptions.
8. Removal and refresh require a trustworthy restoration basis. Protect images, configuration, administration, and retained state, and demonstrate timely reestablishment of required capability.
9. The principles of *Least Functionality*, *Least Privilege*, *Domain Separation*, and *Mediated Access* bound what can happen while a resource remains available. The *Least Persistence* principle complements those restrictions and does not substitute for them.
10. Assurance must demonstrate required availability, effective withdrawal, safe transitions, justified retention, and trustworthy restoration. Maintain that evidence across updates, exceptions, operational change, and recovery.

12. Conclusion

Least persistence makes time an explicit part of trustworthy secure design. A resource can be necessary, correctly separated, narrowly privileged, and properly mediated, yet remain available longer than its purpose justifies. That unnecessary duration creates opportunities for inadvertent use, unauthorized activation, corruption, disclosure, and continued adversary influence. The engineering task is to align effective availability and use with demonstrated need.

NIST's principle provides a practical progression: avoid unnecessary presence and operation, remove resources or power when need ends, use disablement or safe modes where appropriate, and employ dependable mediation where those alternatives are not feasible [1]. The chosen mechanism must preserve required mission and protection behavior. It must also establish the particular temporal property claimed, rather than rely on a general label such as temporary or refreshed.

Engineering for Compromise sharpens the question. If an element becomes hostile while available, what can it accomplish before its interval ends, what influence survives afterward, and which independent mechanisms constrain that influence? Least persistence can reduce exposure and end selected capabilities, but durable effects and compromised dependencies require complementary protections. A replacement is valuable only when the system can establish a trustworthy basis for its return [2].

The principle is consequential for both producers and consumers of computing technologies. Producers should design enforceable lifetimes, safe transitions, bounded renewal, and protected restoration. Consumers should ask which resources remain available between tasks, why they remain necessary, who can extend their availability, and what evidence establishes that access and effects end as intended. These questions apply to conventional computing, operational technology, and AI-enabled systems alike.

A trustworthy secure system provides required capability at the time it is needed, withdraws unjustified exposure when that need ends, and restores necessary capability through trustworthy means. Least persistence turns that expectation into an engineering obligation sustained through the system life cycle.

***“Provide the capability when it is needed. End its exposure when the need ends.
Substantiate both.”***

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. McEvilly, *Engineering Trustworthy Secure Systems*, NIST Special Publication 800-160, Volume 1, Revision 1, National Institute of Standards and Technology, November 2022. <https://doi.org/10.6028/NIST.SP.800-160v1r1>
- [2] R. Ross, *Engineering for Compromise: Designing Systems That Remain Trustworthy Secure Under Adversity*, RONROSSECURE, LLC, August 2026. <https://ronrossecure.com/library>
- [3] R. Ross, V. Pillitteri, R. Graubart, D. Bodeau, and R. McQuaid, *Developing Cyber-Resilient Systems: A Systems Security Engineering Approach*, NIST Special Publication 800-160, Volume 2, Revision 1, National Institute of Standards and Technology, December 2021. <https://doi.org/10.6028/NIST.SP.800-160v2r1>