CANONICAL LIVING OVERVIEW GUIDE

Threat Modeling and Security Architecture

A system-level threat-modeling and security-architecture guide covering assets, data flows, trust boundaries, actors, attack surfaces, abuse cases, STRIDE-style analysis, attack trees, adversary behavior, control selection, residual risk, architecture review, verification, and living-model governance.

PERMANENT ID
MYTHOS-FG-SEC-0001

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Threat Modeling and Security Architecture

Document ID
MYTHOS-FG-SEC-0001

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography
Secondary domain	MYTHOS-SWE; MYTHOS-NET; MYTHOS-AI
Audience	Security architects, network and software engineers, product teams, cloud teams, risk owners, and technical reviewers.
Prerequisites	General system architecture, networking, identity, application, and security fundamentals.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Create a threat model tied to real architecture, assets, trust boundaries, and business impact.

Use multiple modeling techniques without mistaking a checklist for complete analysis.

Translate threats into preventative, detective, responsive, and recovery controls with owners and evidence.

Validate security claims through architecture review, testing, telemetry, and residual-risk decisions.

Maintain threat models as systems, dependencies, adversaries, and assumptions change.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Security objectives, assets, and decision context

Define what must be protected, why it matters, and who owns the risk.

Chapter operating position
Define what must be protected, why it matters, and who owns the risk.

1.1 Mission and business impact

Mission and business impact must be engineered as an observable mechanism rather than a product label or checklist item. Connect confidentiality, integrity, availability, safety, privacy, trust, legal, financial, and operational outcomes. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for mission and business impact.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating mission and business impact as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for mission and business impact.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

1.2 Assets and security properties

Assets and security properties must be engineered as an observable mechanism rather than a product label or checklist item. Identify data, identities, credentials, code, models, infrastructure, communications, decisions, logs, and physical processes. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for assets and security properties.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
threat:
  id: TM-042
  asset: signing-key
  attacker_goal: publish-unauthorized-artifact
  path:
    - compromise-build-identity
    - access-signing-service
    - bypass-release-approval
  controls:
    preventive: [workload-identity, threshold-signing, separation-of-duties]
    detective: [signed-build-provenance, anomalous-signing-alert]
    recovery: [revoke-key, withdraw-release, rebuild-trust]
  evidence:
    - architecture-boundary
    - control-test
    - residual-risk-owner
```

Failure patterns

- Treating assets and security properties as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for assets and security properties.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

1.3 Stakeholders and risk authority

Stakeholders and risk authority must be engineered as an observable mechanism rather than a product label or checklist item. Name system owners, data owners, security, privacy, operations, safety, vendors, users, and acceptance authority. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for stakeholders and risk authority.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating stakeholders and risk authority as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for stakeholders and risk authority.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

1.4 Scope, assumptions, and exclusions

Scope, assumptions, and exclusions must be engineered as an observable mechanism rather than a product label or checklist item. Define lifecycle, environments, dependencies, attacker access, trust assumptions, exclusions, and review triggers. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for scope, assumptions, and exclusions.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating scope, assumptions, and exclusions as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for scope, assumptions, and exclusions.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 01 laboratory and review

Practical laboratory
Select a representative system and produce a security-objective and asset register with impact, owner, lifecycle, and evidence requirements.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Extend the model to autonomous agents, synthetic identities, model weights, cyber-physical safety, and quantum-era confidentiality.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Architecture decomposition and trust boundaries

Model the system in enough detail to expose authority transitions and attack paths.

Chapter operating position
Model the system in enough detail to expose authority transitions and attack paths.

2.1 Components, services, and dependencies

Components, services, and dependencies must be engineered as an observable mechanism rather than a product label or checklist item. Identify endpoints, networks, APIs, queues, databases, clouds, third parties, build systems, models, operators, and recovery systems. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for components, services, and dependencies.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating components, services, and dependencies as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for components, services, and dependencies.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

2.2 Data and control flows

Data and control flows must be engineered as an observable mechanism rather than a product label or checklist item. Trace create, read, update, delete, transform, sign, encrypt, approve, execute, replicate, retain, and destroy operations. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for data and control flows.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
threat:
  id: TM-042
  asset: signing-key
  attacker_goal: publish-unauthorized-artifact
  path:
    - compromise-build-identity
    - access-signing-service
    - bypass-release-approval
  controls:
    preventive: [workload-identity, threshold-signing, separation-of-duties]
    detective: [signed-build-provenance, anomalous-signing-alert]
    recovery: [revoke-key, withdraw-release, rebuild-trust]
  evidence:
    - architecture-boundary
    - control-test
    - residual-risk-owner
```

Failure patterns

- Treating data and control flows as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for data and control flows.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

2.3 Trust boundaries and privilege transitions

Trust boundaries and privilege transitions must be engineered as an observable mechanism rather than a product label or checklist item. Mark changes in identity, authority, network, tenant, process, hardware, organization, location, and cryptographic protection. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for trust boundaries and privilege transitions.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating trust boundaries and privilege transitions as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.

- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for trust boundaries and privilege transitions.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

2.4 Deployment and lifecycle views

Deployment and lifecycle views must be engineered as an observable mechanism rather than a product label or checklist item. Model development, build, release, runtime, administration, incident, backup, recovery, decommission, and supply chain. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for deployment and lifecycle views.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating deployment and lifecycle views as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for deployment and lifecycle views.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 02 laboratory and review

Practical laboratory
Create logical, deployment, and data-flow views; label every trust boundary and identify one hidden administrative or supply-chain path.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore machine-readable architecture graphs, semantic trust boundaries, digital twins, and continuously updated threat models.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Threat actors, capabilities, and attack surfaces

Model plausible adversaries and paths without assuming equal likelihood or consequence.

Chapter operating position
Model plausible adversaries and paths without assuming equal likelihood or consequence.

3.1 Actor objectives and access

Actor objectives and access must be engineered as an observable mechanism rather than a product label or checklist item. Describe criminal, state, insider, partner, operator, tenant, user, automated, supply-chain, and accidental actors. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for actor objectives and access.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating actor objectives and access as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for actor objectives and access.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

3.2 Capabilities and constraints

Capabilities and constraints must be engineered as an observable mechanism rather than a product label or checklist item. Estimate credentials, network position, code execution, physical access, resources, persistence, stealth, knowledge, and time. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for capabilities and constraints.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
threat:
  id: TM-042
  asset: signing-key
  attacker_goal: publish-unauthorized-artifact
  path:
    - compromise-build-identity
    - access-signing-service
    - bypass-release-approval
  controls:
    preventive: [workload-identity, threshold-signing, separation-of-duties]
    detective: [signed-build-provenance, anomalous-signing-alert]
    recovery: [revoke-key, withdraw-release, rebuild-trust]
  evidence:
    - architecture-boundary
    - control-test
    - residual-risk-owner
```

Failure patterns

- Treating capabilities and constraints as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for capabilities and constraints.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

3.3 Attack-surface inventory

Attack-surface inventory must be engineered as an observable mechanism rather than a product label or checklist item. Enumerate interfaces, protocols, identities, management, APIs, files, imports, plugins, devices, models, logs, backups, and human workflows. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for attack-surface inventory.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating attack-surface inventory as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for attack-surface inventory.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

3.4 Adversary behavior knowledge

Adversary behavior knowledge must be engineered as an observable mechanism rather than a product label or checklist item. Use ATT&CK and other evidence to inform behavior while avoiding unsupported one-to-one control claims. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for adversary behavior knowledge.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating adversary behavior knowledge as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for adversary behavior knowledge.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 03 laboratory and review

Practical laboratory
Build three actor profiles and map plausible entry, execution, persistence, movement, collection, impact, and recovery-interference paths.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Model AI-assisted adversaries, autonomous exploitation, model manipulation, hardware attacks, and photonic or quantum infrastructure exposure.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Threat-identification methods

Use complementary methods to find design-specific threats and abuse paths.

Chapter operating position
Use complementary methods to find design-specific threats and abuse paths.

4.1 STRIDE-style analysis

STRIDE-style analysis must be engineered as an observable mechanism rather than a product label or checklist item. Examine spoofing, tampering, repudiation, disclosure, denial of service, and elevation of privilege by element and flow. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for stride-style analysis.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating stride-style analysis as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for stride-style analysis.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

4.2 Attack trees and attack graphs

Attack trees and attack graphs must be engineered as an observable mechanism rather than a product label or checklist item. Decompose goals into alternative and dependent steps, prerequisites, cost, detection, and control points. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for attack trees and attack graphs.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
threat:
  id: TM-042
  asset: signing-key
  attacker_goal: publish-unauthorized-artifact
  path:
    - compromise-build-identity
    - access-signing-service
    - bypass-release-approval
  controls:
    preventive: [workload-identity, threshold-signing, separation-of-duties]
    detective: [signed-build-provenance, anomalous-signing-alert]
    recovery: [revoke-key, withdraw-release, rebuild-trust]
  evidence:
    - architecture-boundary
    - control-test
    - residual-risk-owner
```

Failure patterns

- Treating attack trees and attack graphs as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for attack trees and attack graphs.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

4.3 Abuse and misuse cases

Abuse and misuse cases must be engineered as an observable mechanism rather than a product label or checklist item. Describe harmful user, operator, API, workflow, model, and automation behaviors end to end. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for abuse and misuse cases.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating abuse and misuse cases as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for abuse and misuse cases.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

4.4 Data-centric and privacy-focused modeling

Data-centric and privacy-focused modeling must be engineered as an observable mechanism rather than a product label or checklist item. Follow sensitive data through collection, inference, transformation, access, retention, egress, and deletion. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for data-centric and privacy-focused modeling.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating data-centric and privacy-focused modeling as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for data-centric and privacy-focused modeling.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 04 laboratory and review

Practical laboratory
Apply three methods to the same architecture and compare what each found, missed, duplicated, or expressed differently.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Evaluate formal attack graphs, automated path generation, model checking, and evidence-calibrated AI assistance.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Control architecture and defense composition

Turn threats into coherent controls rather than isolated product features.

Chapter operating position
Turn threats into coherent controls rather than isolated product features.

5.1 Preventative and limiting controls

Preventative and limiting controls must be engineered as an observable mechanism rather than a product label or checklist item. Use secure design, minimization, isolation, authentication, authorization, encryption, validation, safe defaults, and least privilege. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for preventative and limiting controls.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating preventative and limiting controls as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for preventative and limiting controls.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

5.2 Detective and evidence controls

Detective and evidence controls must be engineered as an observable mechanism rather than a product label or checklist item. Define telemetry, integrity, correlation, alerting, human review, provenance, and tamper-resistant records. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for detective and evidence controls.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
threat:
  id: TM-042
  asset: signing-key
  attacker_goal: publish-unauthorized-artifact
  path:
    - compromise-build-identity
    - access-signing-service
    - bypass-release-approval
  controls:
    preventive: [workload-identity, threshold-signing, separation-of-duties]
    detective: [signed-build-provenance, anomalous-signing-alert]
    recovery: [revoke-key, withdraw-release, rebuild-trust]
  evidence:
    - architecture-boundary
    - control-test
    - residual-risk-owner
```

Failure patterns

- Treating detective and evidence controls as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for detective and evidence controls.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

5.3 Response and recovery controls

Response and recovery controls must be engineered as an observable mechanism rather than a product label or checklist item. Design containment, revocation, fail-safe states, rollback, backup, rebuild, communications, and lessons learned. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for response and recovery controls.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating response and recovery controls as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for response and recovery controls.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

5.4 Control dependencies and bypass

Control dependencies and bypass must be engineered as an observable mechanism rather than a product label or checklist item. Identify prerequisite identity, time, trust stores, management planes, updates, operators, and shared infrastructure. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for control dependencies and bypass.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating control dependencies and bypass as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for control dependencies and bypass.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 05 laboratory and review

Practical laboratory
Choose controls for ten threats, identify dependencies and bypass paths, and define examine, interview, test, and measure evidence.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore adaptive controls, confidential computing, post-quantum crypto agility, self-healing systems, and verifiable policy execution.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Risk analysis, prioritization, and residual risk

Make explicit decisions about consequence, plausibility, uncertainty, and treatment.

Chapter operating position
Make explicit decisions about consequence, plausibility, uncertainty, and treatment.

6.1 Likelihood, impact, and uncertainty

Likelihood, impact, and uncertainty must be engineered as an observable mechanism rather than a product label or checklist item. Use evidence, exposure, capability, controls, detectability, recovery, and confidence rather than false precision. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for likelihood, impact, and uncertainty.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating likelihood, impact, and uncertainty as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for likelihood, impact, and uncertainty.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

6.2 Risk chains and aggregation

Risk chains and aggregation must be engineered as an observable mechanism rather than a product label or checklist item. Recognize how weaknesses combine into high-impact paths and how common dependencies create systemic risk. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for risk chains and aggregation.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
threat:
  id: TM-042
  asset: signing-key
  attacker_goal: publish-unauthorized-artifact
  path:
    - compromise-build-identity
    - access-signing-service
    - bypass-release-approval
  controls:
    preventive: [workload-identity, threshold-signing, separation-of-duties]
    detective: [signed-build-provenance, anomalous-signing-alert]
    recovery: [revoke-key, withdraw-release, rebuild-trust]
  evidence:
    - architecture-boundary
    - control-test
    - residual-risk-owner
```

Failure patterns

- Treating risk chains and aggregation as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for risk chains and aggregation.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

6.3 Treatment and acceptance

Treatment and acceptance must be engineered as an observable mechanism rather than a product label or checklist item. Choose avoid, reduce, transfer, monitor, or accept with authority, duration, evidence, and review triggers. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for treatment and acceptance.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating treatment and acceptance as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for treatment and acceptance.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

6.4 Safety, privacy, and mission interaction

Safety, privacy, and mission interaction must be engineered as an observable mechanism rather than a product label or checklist item. Avoid treatments that create unacceptable safety, privacy, usability, availability, or operational consequences. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for safety, privacy, and mission interaction.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating safety, privacy, and mission interaction as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for safety, privacy, and mission interaction.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 06 laboratory and review

Practical laboratory
Prioritize modeled threats using a transparent method, compare with a second method, and document disagreements and assumptions.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Evaluate probabilistic attack graphs, Bayesian updating, continuous risk telemetry, and simulation-based estimates.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Verification, review, and security-case evidence

Prove that the implemented system supports its security claims.

Chapter operating position
Prove that the implemented system supports its security claims.

7.1 Architecture review and traceability

Architecture review and traceability must be engineered as an observable mechanism rather than a product label or checklist item. Map requirements, threats, controls, components, tests, findings, exceptions, and residual risk. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for architecture review and traceability.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating architecture review and traceability as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for architecture review and traceability.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

7.2 Security testing and adversarial validation

Security testing and adversarial validation must be engineered as an observable mechanism rather than a product label or checklist item. Use unit, integration, configuration, protocol, abuse, penetration, red-team, failure, and recovery testing. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for security testing and adversarial validation.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
threat:
  id: TM-042
  asset: signing-key
  attacker_goal: publish-unauthorized-artifact
  path:
    - compromise-build-identity
    - access-signing-service
    - bypass-release-approval
  controls:
    preventive: [workload-identity, threshold-signing, separation-of-duties]
    detective: [signed-build-provenance, anomalous-signing-alert]
    recovery: [revoke-key, withdraw-release, rebuild-trust]
  evidence:
    - architecture-boundary
    - control-test
    - residual-risk-owner
```

Failure patterns

- Treating security testing and adversarial validation as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for security testing and adversarial validation.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

7.3 Operational evidence

Operational evidence must be engineered as an observable mechanism rather than a product label or checklist item. Confirm identity, policy, telemetry, alerting, revocation, backup, incident, and recovery behavior. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for operational evidence.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating operational evidence as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for operational evidence.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

7.4 Security case and assurance argument

Security case and assurance argument must be engineered as an observable mechanism rather than a product label or checklist item. State claims, evidence, assumptions, defeaters, limitations, review status, and accountable approval. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for security case and assurance argument.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating security case and assurance argument as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for security case and assurance argument.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 07 laboratory and review

Practical laboratory

Construct a security case for one critical claim and attempt to disprove it through review, testing, and failure injection.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore machine-verifiable assurance cases, signed evidence bundles, continuous control validation, and autonomous red/blue simulation.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Living threat-model governance

Keep the model aligned with changing architecture, dependencies, threats, and evidence.

Chapter operating position
Keep the model aligned with changing architecture, dependencies, threats, and evidence.

8.1 Change triggers and lifecycle integration

Change triggers and lifecycle integration must be engineered as an observable mechanism rather than a product label or checklist item. Trigger review for new features, interfaces, data, dependencies, vendors, models, incidents, vulnerabilities, and environments. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for change triggers and lifecycle integration.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating change triggers and lifecycle integration as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for change triggers and lifecycle integration.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

8.2 Model ownership and collaboration

Model ownership and collaboration must be engineered as an observable mechanism rather than a product label or checklist item. Define security, engineering, product, operations, risk, privacy, safety, and executive responsibilities. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for model ownership and collaboration.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
threat:
  id: TM-042
  asset: signing-key
  attacker_goal: publish-unauthorized-artifact
  path:
    - compromise-build-identity
    - access-signing-service
    - bypass-release-approval
  controls:
    preventive: [workload-identity, threshold-signing, separation-of-duties]
    detective: [signed-build-provenance, anomalous-signing-alert]
    recovery: [revoke-key, withdraw-release, rebuild-trust]
  evidence:
    - architecture-boundary
    - control-test
    - residual-risk-owner
```

Failure patterns

- Treating model ownership and collaboration as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for model ownership and collaboration.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

8.3 Tooling and machine-readable models

Tooling and machine-readable models must be engineered as an observable mechanism rather than a product label or checklist item. Use diagrams, structured threats, graphs, issue trackers, code annotations, tests, evidence, and release gates. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for tooling and machine-readable models.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating tooling and machine-readable models as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for tooling and machine-readable models.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

8.4 Metrics and program maturity

Metrics and program maturity must be engineered as an observable mechanism rather than a product label or checklist item. Measure coverage, review age, unresolved critical paths, validation, recurrence, design changes, and decision quality. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the mission, asset, architecture, trust boundary, actor, attack path, control, evidence, and residual-risk decision chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for metrics and program maturity.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating metrics and program maturity as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for metrics and program maturity.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 08 laboratory and review

Practical laboratory
Integrate a threat model into a software or infrastructure change workflow with review, traceability, tests, approval, and revision history.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Develop a governed threat-model agent that proposes threats and tests but must cite architecture evidence.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. NIST - Cybersecurity Framework 2.0
Role: Risk-governance and cybersecurity outcome framework.
Verified: 2026-07-26
Official source: https://www.nist.gov/cyberframework

02. NIST - SP 800-53 Rev. 5 - Security and Privacy Controls
Role: Control catalog for security and privacy architecture and assurance.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final

03. NIST - Draft SP 800-154 - Guide to Data-Centric System Threat Modeling
Role: Draft threat-modeling methodology used with explicit status limitations.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/154/ipd

04. MITRE - MITRE ATT&CK
Role: Knowledge base of adversary tactics and techniques based on real-world observations.
Verified: 2026-07-26
Official source: https://attack.mitre.org/

05. MITRE - MITRE D3FEND
Role: Knowledge graph of defensive cybersecurity techniques.
Verified: 2026-07-26
Official source: https://d3fend.mitre.org/

06. OWASP Foundation - OWASP Threat Modeling
Role: Community guidance for practical application and software threat modeling.
Verified: 2026-07-26
Official source: https://owasp.org/www-community/Threat_Modeling

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography
Secondary domain	MYTHOS-SWE; MYTHOS-NET; MYTHOS-AI
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	threat modeling, security architecture, STRIDE, attack trees, ATT&CK, risk, trust boundaries, security case, assurance, architecture review
Canonical route	/library/field-guides/mythos-fg-sec-0001/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.