



CANONICAL LIVING OVERVIEW GUIDE

802.1X, RADIUS, NAC, and Device Profiling

A full identity-aware access guide covering 802.1X, EAP, RADIUS, certificates, wired and wireless enforcement, profiling, posture, guest and IoT onboarding, dynamic policy, high availability, troubleshooting, telemetry, and zero-trust evolution.

PERMANENT PUBLICATION IDENTITY

802.1X, RADIUS, NAC, and Device Profiling

Document ID

MYTHOS-FG-NET-0014

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

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-NET - Network Engineering & Architecture
Secondary domain	MYTHOS-SEC; MYTHOS-DAT
Audience	Network security, NAC, identity, wireless, campus, PKI, endpoint, and operations engineers.
Prerequisites	Switching, wireless, IP, authentication, and certificate fundamentals.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

- Trace supplicant, authenticator, RADIUS, EAP, certificate, policy, and enforcement state.
- Design EAP-TLS and certificate lifecycle for users, devices, and machine identity.
- Use profiling and fallback methods without converting uncertainty into implicit trust.
- Operate dynamic segmentation, posture, guest, and IoT onboarding with resilient dependencies.
- Diagnose failures through packet, endpoint, switch, AP, RADIUS, directory, PKI, and policy evidence.

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

How to use this guide	3
Contents	3
Chapter 01 - Access-control architecture	5
Chapter 01 laboratory and review	10
Chapter 02 - EAP methods and secure identity	11
Chapter 02 laboratory and review	16
Chapter 03 - RADIUS engineering	17
Chapter 03 laboratory and review	22
Chapter 04 - Certificates, PKI, and lifecycle	23
Chapter 04 laboratory and review	28
Chapter 05 - NAC policy and enforcement	29
Chapter 05 laboratory and review	34

Chapter 06 - Profiling, posture, and non-802.1X devices	35
Chapter 06 laboratory and review	40
Chapter 07 - High availability and operations	41
Chapter 07 laboratory and review	46
Chapter 08 - Troubleshooting and assurance	47
Chapter 08 laboratory and review	52
Integrated learning roadmap	53
Source authority register	54
Limitations, freshness, and revision control	55

CHAPTER 01

Access-control architecture

Understand the entities, trust boundaries, protocols, and decisions in identity-aware access.

Chapter operating position

Understand the entities, trust boundaries, protocols, and decisions in identity-aware access.



1.1 Supplicant, authenticator, and server

Supplicant, authenticator, and server must be treated as an engineering mechanism rather than a product checkbox. Separate endpoint identity exchange, port or WLAN enforcement, RADIUS transport, and policy authority. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for supplicant, authenticator, and server.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating supplicant, authenticator, and server as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for supplicant, authenticator, and server.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

1.2 Controlled and uncontrolled ports

Controlled and uncontrolled ports must be treated as an engineering mechanism rather than a product checkbox. Understand EAPOL availability, authorization state, data access, fallback, and reauthentication. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for controlled and uncontrolled ports.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Implementation sketch

```
access_policy:
  identity: managed-device-certificate
  eap_method: eap-tls-1.3
  conditions:
    - certificate_chain_valid
    - device_record_active
    - posture_compliant
  authorization:
    segment: corp-managed
    acl: least-privilege
  fallback:
    mode: restricted
    expires_minutes: 30
evidence:
  - eapol_exchange
  - radius_result_and_attributes
  - certificate_validation
  - applied_port_policy
  - accounting_session
```

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for controlled and uncontrolled ports.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.



1.3 Authentication, authorization, and accounting

Authentication, authorization, and accounting must be treated as an engineering mechanism rather than a product checkbox. Distinguish proof of identity, policy result, session attributes, and retained audit. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and

partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for authentication, authorization, and accounting.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating authentication, authorization, and accounting as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for authentication, authorization, and accounting.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.



1.4 Wired, wireless, VPN, and administrative access

Wired, wireless, VPN, and administrative access must be treated as an engineering mechanism rather than a product checkbox. Apply common identity concepts while recognizing different enforcement and transport behavior. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for wired, wireless, vpn, and administrative access.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating wired, wireless, vpn, and administrative access as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for wired, wireless, vpn, and administrative access.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

Chapter 01 laboratory and review

Practical laboratory

Trace a complete wired and wireless authentication, identifying every message, state transition, policy input, and enforcement action.

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 architecture to workload identity, agent identity, continuous authorization, and device attestation.

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

EAP methods and secure identity

Choose authentication methods by cryptographic assurance, provisioning, user experience, and operational risk.

Chapter operating position

Choose authentication methods by cryptographic assurance, provisioning, user experience, and operational risk.

2.1 EAP framework

EAP framework must be treated as an engineering mechanism rather than a product checkbox. Understand identity exchange, method negotiation, success and failure, keying material, fragmentation, and retransmission. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, and acceptance criteria for eap framework.
- Model the expected state and traffic path before implementation or troubleshooting.
- Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.
- Test normal, denied, degraded, failed, recovery, upgrade, and rollback conditions.
- Retain topology, configuration, packet, telemetry, log, identity, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded concurrency, and reversible changes.

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for eap framework.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

2.2 EAP-TLS with TLS 1.3

EAP-TLS with TLS 1.3 must be treated as an engineering mechanism rather than a product checkbox. Use mutual certificates, trust chains, identities, revocation, server validation, resumption, and key derivation. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for eap-tls with tls 1.3.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Implementation sketch

```
access_policy:
  identity: managed-device-certificate
  eap_method: eap-tls-1.3
  conditions:
    - certificate_chain_valid
    - device_record_active
    - posture_compliant
  authorization:
    segment: corp-managed
    acl: least-privilege
  fallback:
    mode: restricted
    expires_minutes: 30
evidence:
  - eapol_exchange
  - radius_result_and_attributes
  - certificate_validation
  - applied_port_policy
  - accounting_session
```

Failure patterns

Treating eap-tls with tls 1.3 as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for eap-tls with tls 1.3.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.



2.3 Tunneled methods

Tunneled methods must be treated as an engineering mechanism rather than a product checkbox. Understand PEAP, EAP-TTLS, TEAP, inner methods, credential exposure, TLS 1.3 considerations, and migration. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and

partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, and acceptance criteria for tunneled methods.
- Model the expected state and traffic path before implementation or troubleshooting.
- Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.
- Test normal, denied, degraded, failed, recovery, upgrade, and rollback conditions.
- Retain topology, configuration, packet, telemetry, log, identity, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded concurrency, and reversible changes.

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for tunneled methods.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

2.4 Method selection and deprecation

Method selection and deprecation must be treated as an engineering mechanism rather than a product checkbox. Avoid weak credentials, insecure server validation, anonymous ambiguity, and unsupported legacy methods. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for method selection and deprecation.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating method selection and deprecation as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for method selection and deprecation.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

Chapter 02 laboratory and review

Practical laboratory

Capture EAP-TLS and one tunneled EAP method, validate server and client certificates, then break trust, identity, time, revocation, and TLS settings.

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 post-quantum TLS, hardware-bound credentials, passkey-linked enrollment, and privacy-preserving identities.

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

RADIUS engineering

Operate the AAA transport, attributes, policy exchange, and accounting reliably.

Chapter operating position

Operate the AAA transport, attributes, policy exchange, and accounting reliably.

3.1 Requests, responses, and shared trust

Requests, responses, and shared trust must be treated as an engineering mechanism rather than a product checkbox. Trace Access-Request, Challenge, Accept, Reject, authenticators, shared secrets, source identity, and retransmission. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for requests, responses, and shared trust.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating requests, responses, and shared trust as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for requests, responses, and shared trust.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

3.2 Attributes and policy

Attributes and policy must be treated as an engineering mechanism rather than a product checkbox. Use standard, vendor, tunnel, filter, session, EAP, and accounting attributes with explicit ownership. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for attributes and policy.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Implementation sketch

```
access_policy:
  identity: managed-device-certificate
  eap_method: eap-tls-1.3
  conditions:
    - certificate_chain_valid
    - device_record_active
    - posture_compliant
  authorization:
    segment: corp-managed
    acl: least-privilege
  fallback:
    mode: restricted
    expires_minutes: 30
evidence:
  - eapol_exchange
  - radius_result_and_attributes
  - certificate_validation
  - applied_port_policy
  - accounting_session
```

Failure patterns

Treating attributes and policy as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for attributes and policy.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.



3.3 Proxy and federation

Proxy and federation must be treated as an engineering mechanism rather than a product checkbox. Design realms, proxy chains, eduroam-like federation, trust, routing, loops, privacy, and troubleshooting. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and

partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, and acceptance criteria for proxy and federation.
- Model the expected state and traffic path before implementation or troubleshooting.
- Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.
- Test normal, denied, degraded, failed, recovery, upgrade, and rollback conditions.
- Retain topology, configuration, packet, telemetry, log, identity, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded concurrency, and reversible changes.

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for proxy and federation.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

3.4 Accounting and change of authorization

Accounting and change of authorization must be treated as an engineering mechanism rather than a product checkbox. Track start, stop, interim, session identifiers, disconnect, policy updates, and failure behavior. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for accounting and change of authorization.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating accounting and change of authorization as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for accounting and change of authorization.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

Chapter 03 laboratory and review

Practical laboratory

Build redundant RADIUS service, test timeout, reject, challenge, proxy, accounting loss, and change-of-authorization behavior.

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 RADIUS over TLS, cryptographic agility, event-driven authorization, and globally distributed policy services.

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

Certificates, PKI, and lifecycle

Make certificate identity deployable, renewable, revocable, and recoverable at enterprise scale.

Chapter operating position

Make certificate identity deployable, renewable, revocable, and recoverable at enterprise scale.

4.1 Certificate profiles and identity

Certificate profiles and identity must be treated as an engineering mechanism rather than a product checkbox. Define subjects, SANs, EKUs, key use, issuer, trust, device identity, user identity, and mapping. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for certificate profiles and identity.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating certificate profiles and identity as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for certificate profiles and identity.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

4.2 Enrollment and provisioning

Enrollment and provisioning must be treated as an engineering mechanism rather than a product checkbox. Use MDM, GPO, SCEP, EST, ACME-like workflows, hardware keys, secure bootstrap, and recovery. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for enrollment and provisioning.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Implementation sketch

```
access_policy:
  identity: managed-device-certificate
  eap_method: eap-tls-1.3
  conditions:
    - certificate_chain_valid
    - device_record_active
    - posture_compliant
  authorization:
    segment: corp-managed
    acl: least-privilege
  fallback:
    mode: restricted
    expires_minutes: 30
evidence:
  - eapol_exchange
  - radius_result_and_attributes
  - certificate_validation
  - applied_port_policy
  - accounting_session
```

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for enrollment and provisioning.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.



4.3 Revocation and validation

Revocation and validation must be treated as an engineering mechanism rather than a product checkbox. Operate CRL, OCSP, short-lived certificates, cache, soft fail, hard fail, clock, and outage policy. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and

partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for revocation and validation.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating revocation and validation as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for revocation and validation.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

4.4 Renewal and migration

Renewal and migration must be treated as an engineering mechanism rather than a product checkbox. Plan overlap, trust-anchor changes, algorithm agility, certificate replacement, and stranded devices. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for renewal and migration.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating renewal and migration as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for renewal and migration.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

Chapter 04 laboratory and review

Practical laboratory

Create a certificate lifecycle from enrollment to renewal and revocation. Simulate CA, CRL, OCSP, time, and trust-store failures.

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

Assess post-quantum certificates, hybrid chains, device-attested keys, and automated crypto-agility.

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

NAC policy and enforcement

Convert identity, device, posture, location, and risk into least-privilege network access.

Chapter operating position

Convert identity, device, posture, location, and risk into least-privilege network access.

5.1 Policy inputs and precedence

Policy inputs and precedence must be treated as an engineering mechanism rather than a product checkbox. Use identity, group, certificate, device class, posture, site, SSID, port, time, and threat signals. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for policy inputs and precedence.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating policy inputs and precedence as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for policy inputs and precedence.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

5.2 Dynamic authorization

Dynamic authorization must be treated as an engineering mechanism rather than a product checkbox. Apply VLAN, ACL, security group, VRF, SGT-like tag, firewall, quarantine, and service-chain actions. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for dynamic authorization.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Implementation sketch

```
access_policy:
  identity: managed-device-certificate
  eap_method: eap-tls-1.3
  conditions:
    - certificate_chain_valid
    - device_record_active
    - posture_compliant
  authorization:
    segment: corp-managed
    acl: least-privilege
  fallback:
    mode: restricted
    expires_minutes: 30
evidence:
  - eapol_exchange
  - radius_result_and_attributes
  - certificate_validation
  - applied_port_policy
  - accounting_session
```

Failure patterns

Treating dynamic authorization as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for dynamic authorization.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.



5.3 Reauthentication and continuous evaluation

Reauthentication and continuous evaluation must be treated as an engineering mechanism rather than a product checkbox. Handle timers, posture changes, threat events, role changes, movement, and session termination. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and

partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for reauthentication and continuous evaluation.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating reauthentication and continuous evaluation as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for reauthentication and continuous evaluation.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

5.4 Fail-open, fail-closed, and critical access

Fail-open, fail-closed, and critical access must be treated as an engineering mechanism rather than a product checkbox. Define behavior during NAC, PKI, directory, WAN, or controller failure by service criticality. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for fail-open, fail-closed, and critical access.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating fail-open, fail-closed, and critical access as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for fail-open, fail-closed, and critical access.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

Chapter 05 laboratory and review

Practical laboratory

Design policy for employees, contractors, guests, IoT, OT, printers, phones, and privileged admins. Test identity and posture changes in-session.

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 risk-adaptive access, zero-trust policy engines, workload and agent access, and verifiable authorization decisions.

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

Profiling, posture, and non-802.1X devices

Classify uncertain devices without mistaking heuristics for authenticated identity.

Chapter operating position

Classify uncertain devices without mistaking heuristics for authenticated identity.



6.1 Profiling evidence

Profiling evidence must be treated as an engineering mechanism rather than a product checkbox. Use DHCP, LLDP, CDP, RADIUS, MAC OUI, HTTP, TLS, SNMP, flow, and active probes with confidence. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, and acceptance criteria for profiling evidence.
- Model the expected state and traffic path before implementation or troubleshooting.
- Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.
- Test normal, denied, degraded, failed, recovery, upgrade, and rollback conditions.
- Retain topology, configuration, packet, telemetry, log, identity, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded concurrency, and reversible changes.

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for profiling evidence.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

6.2 MAB and exception handling

MAB and exception handling must be treated as an engineering mechanism rather than a product checkbox. Treat MAC-based bypass as weak identification with restricted policy, ownership, review, and expiry. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for mab and exception handling.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Implementation sketch

```
access_policy:
  identity: managed-device-certificate
  eap_method: eap-tls-1.3
  conditions:
    - certificate_chain_valid
    - device_record_active
    - posture_compliant
  authorization:
    segment: corp-managed
    acl: least-privilege
  fallback:
    mode: restricted
    expires_minutes: 30
evidence:
  - eapol_exchange
  - radius_result_and_attributes
  - certificate_validation
  - applied_port_policy
  - accounting_session
```

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for mab and exception handling.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.



6.3 Posture assessment

Posture assessment must be treated as an engineering mechanism rather than a product checkbox. Evaluate managed state, certificates, security software, patch, encryption, compliance, and privacy constraints. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and

partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, and acceptance criteria for posture assessment.
- Model the expected state and traffic path before implementation or troubleshooting.
- Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.
- Test normal, denied, degraded, failed, recovery, upgrade, and rollback conditions.
- Retain topology, configuration, packet, telemetry, log, identity, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded concurrency, and reversible changes.

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for posture assessment.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

6.4 IoT and OT onboarding

IoT and OT onboarding must be treated as an engineering mechanism rather than a product checkbox. Use inventory, ownership, secure profiling, behavior baselines, segmentation, and compensating controls. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for iot and ot onboarding.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating iot and ot onboarding as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for iot and ot onboarding.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

Chapter 06 laboratory and review

Practical laboratory

Profile known and unknown devices, manipulate their observable attributes, and document confidence, false positives, policy, and expiration.

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 passive behavioral identity, attestation, manufacturer credentials, and zero-touch secure device onboarding.

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

High availability and operations

Keep access available without bypassing policy silently.

Chapter operating position

Keep access available without bypassing policy silently.

7.1 Dependency architecture

Dependency architecture must be treated as an engineering mechanism rather than a product checkbox. Map RADIUS, directory, CA, revocation, DNS, DHCP, time, MDM, posture, controller, switch, and WAN dependencies. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, and acceptance criteria for dependency architecture.
- Model the expected state and traffic path before implementation or troubleshooting.
- Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.
- Test normal, denied, degraded, failed, recovery, upgrade, and rollback conditions.
- Retain topology, configuration, packet, telemetry, log, identity, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded concurrency, and reversible changes.

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for dependency architecture.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

7.2 Redundancy and state

Redundancy and state must be treated as an engineering mechanism rather than a product checkbox. Design node, site, database, certificate, and policy redundancy with deterministic failover. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for redundancy and state.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Implementation sketch

```
access_policy:
  identity: managed-device-certificate
  eap_method: eap-tls-1.3
  conditions:
    - certificate_chain_valid
    - device_record_active
    - posture_compliant
  authorization:
    segment: corp-managed
    acl: least-privilege
  fallback:
    mode: restricted
    expires_minutes: 30
evidence:
  - eapol_exchange
  - radius_result_and_attributes
  - certificate_validation
  - applied_port_policy
  - accounting_session
```

Failure patterns

Treating redundancy and state as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for redundancy and state.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

7.3 Telemetry and service objectives

Telemetry and service objectives must be treated as an engineering mechanism rather than a product checkbox. Track authentication success, latency, failure reason, EAP method, endpoint class, policy, accounting, and change. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and

partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for telemetry and service objectives.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating telemetry and service objectives as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for telemetry and service objectives.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

7.4 Change and lifecycle

Change and lifecycle must be treated as an engineering mechanism rather than a product checkbox. Validate policy, certificates, device software, EAP support, switch templates, and phased rollout. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for change and lifecycle.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating change and lifecycle as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for change and lifecycle.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

Chapter 07 laboratory and review

Practical laboratory

Fail each core dependency independently and measure which endpoint populations retain, lose, or degrade access. Validate recovery and queued accounting.

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 edge-resident authorization, local-first policy, sovereign identity, and distributed decision 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.

CHAPTER 08

Troubleshooting and assurance

Follow a deterministic workflow from endpoint configuration through EAP, RADIUS, policy, and forwarding.

Chapter operating position

Follow a deterministic workflow from endpoint configuration through EAP, RADIUS, policy, and forwarding.

8.1 Endpoint and supplicant

Endpoint and supplicant must be treated as an engineering mechanism rather than a product checkbox. Check adapter, service, profile, identity, certificate, trust, time, credentials, driver, and EAP state. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, and acceptance criteria for endpoint and supplicant.
- Model the expected state and traffic path before implementation or troubleshooting.
- Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.
- Test normal, denied, degraded, failed, recovery, upgrade, and rollback conditions.
- Retain topology, configuration, packet, telemetry, log, identity, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded concurrency, and reversible changes.

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for endpoint and supplicant.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

8.2 Authenticator and transport

Authenticator and transport must be treated as an engineering mechanism rather than a product checkbox. Check port or WLAN state, EAPOL, VLAN, RADIUS source, reachability, shared secret, timers, and MTU. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for authenticator and transport.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Implementation sketch

```
access_policy:
  identity: managed-device-certificate
  eap_method: eap-tls-1.3
  conditions:
    - certificate_chain_valid
    - device_record_active
    - posture_compliant
  authorization:
    segment: corp-managed
    acl: least-privilege
  fallback:
    mode: restricted
    expires_minutes: 30
evidence:
  - eapol_exchange
  - radius_result_and_attributes
  - certificate_validation
  - applied_port_policy
  - accounting_session
```

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for authenticator and transport.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

8.3 Policy and dependencies

Policy and dependencies must be treated as an engineering mechanism rather than a product checkbox. Check identity lookup, certificate validation, profiling, posture, rule match, attributes, and authorization result. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and

partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, and acceptance criteria for policy and dependencies.
- Model the expected state and traffic path before implementation or troubleshooting.
- Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.
- Test normal, denied, degraded, failed, recovery, upgrade, and rollback conditions.
- Retain topology, configuration, packet, telemetry, log, identity, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded concurrency, and reversible changes.

Failure patterns

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for policy and dependencies.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

8.4 Enforcement and service validation

Enforcement and service validation must be treated as an engineering mechanism rather than a product checkbox. Check applied policy, address assignment, DNS, routing, security, reauth, accounting, and application access. The design should name the authoritative inputs, state transitions, dependencies, limits, ownership, telemetry, expected output, and conditions that make the result unsafe or inconclusive.

The guide traces this subject through the supplicant, EAP, authenticator, RADIUS, certificate, policy, enforcement, and service chain. Configuration, controller health, or a green dashboard is never sufficient proof by itself. The engineer must reconcile intended state with protocol or platform state, installed forwarding or policy, real transactions, failure behavior, and the experience of the affected user, device, service, or application.

Implementation begins with a minimal, known-good baseline and explicit invariants. Introduce one capability or dependency at a time, validate positive and negative cases, inject realistic failures, measure recovery, and preserve before-and-after evidence. Automation must be idempotent, bounded, observable, reversible, and able to stop safely when inputs are stale or outcomes diverge.

Troubleshooting should locate the first divergence from the expected transaction or state machine. Inspect both directions and all authority boundaries; account for caching, eventual consistency, tunnels, load sharing, policy, identity, offload, time, and partial failure. Closure requires causal evidence, restored service, rollback proof, residual-risk disclosure, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, and acceptance criteria for enforcement and service validation.

Model the expected state and traffic path before implementation or troubleshooting.

Validate intent, control state, applied policy or forwarding state, and end-to-end service independently.

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

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

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

Failure patterns

Treating enforcement and service validation as a vendor feature instead of an end-to-end system.

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

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

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

Allowing automation or AI to act on stale, ambiguous, or unauthorized data.

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, topology, models, policy, configuration, ownership, and dependencies for enforcement and service validation.
Observe	Control-plane or platform state, applied policy, forwarding state, telemetry, logs, and endpoint behavior.
Test	Expected traffic, prohibited traffic, dependency loss, partial failure, convergence, recovery, and rollback.
Measure	Availability, reachability, latency, loss, convergence, scale, resource use, change success, and user experience.
Reconcile	Intended state against observed platform, network, security, identity, application, and evidence records.

Chapter 08 laboratory and review

Practical laboratory

Troubleshoot a scenario with certificate trust, time, RADIUS secret, policy-order, DHCP, and CoA faults. Prove each from raw 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

Develop an identity-access digital twin and evidence-citing diagnostic agent with no unilateral authorization authority.

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. IEEE Standards Association - IEEE 802.1X-2020 - Port-Based Network Access Control

Role: Port-based access-control architecture and protocol authority.

Verified: 2026-07-26

Official source: https://standards.ieee.org/standard/802_1X-2020.html

02. IETF / RFC Editor - RFC 2865 - Remote Authentication Dial In User Service

Role: RADIUS authentication and authorization protocol.

Verified: 2026-07-26

Official source: <https://www.rfc-editor.org/info/rfc2865>

03. IETF / RFC Editor - RFC 3748 - Extensible Authentication Protocol

Role: EAP framework authority.

Verified: 2026-07-26

Official source: <https://www.rfc-editor.org/info/rfc3748>

04. IETF / RFC Editor - RFC 9190 - EAP-TLS 1.3

Role: EAP-TLS operation with TLS 1.3.

Verified: 2026-07-26

Official source: <https://www.rfc-editor.org/info/rfc9190>

05. IETF / RFC Editor - RFC 9427 - TLS-Based EAP Types and TLS 1.3

Role: TLS 1.3 guidance for tunneled EAP methods.

Verified: 2026-07-26

Official source: <https://www.rfc-editor.org/info/rfc9427>

06. NIST - NIST SP 800-207 - Zero Trust Architecture

Role: Zero-trust architecture principles and deployment models.

Verified: 2026-07-26

Official source: <https://csrc.nist.gov/pubs/sp/800/207/final>

07. NIST - NIST SP 800-207A - Zero Trust for Cloud-Native Applications

Role: Identity-tier and network-tier policy model for multi-location applications.

Verified: 2026-07-26

Official source: <https://csrc.nist.gov/pubs/sp/800/207/a/final>

08. OpenTelemetry - OpenTelemetry semantic conventions 1.43.0

Role: Current common telemetry naming and semantic conventions.

Verified: 2026-07-26

Official source: <https://opentelemetry.io/docs/specs/semconv/>

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-NET - Network Engineering & Architecture
Secondary domain	MYTHOS-SEC; MYTHOS-DAT
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	802.1X, RADIUS, NAC, EAP-TLS, certificates, profiling, posture, dynamic authorization, zero trust, troubleshooting
Canonical route	/library/field-guides/mythos-fg-net-0014/

Living publication

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