SEAS-8414 CYBER ANALYTICS

Autonomous Remediation and Safety Verification

Plan Engine, Credential Rotation, Firmware Orchestration & Compliance-as-Code

Dr. Mallarapu · Breakwater Security Platform

CHAPTER TAKEAWAY

Let me give you the product statement:

ENRICHMENT VALUE

Every other phase answers "what is the problem?" Phase 12 answers "here is what we did about it, and here is the execution record."

> **Rotated 14 passwords, pushed 3 firmware updates, segmented 6 network zones. IEC 62443: 67% → 91%. Zero downtime.**

- Credential rotator engine
- OTA orchestrator + rollback
- Multi-vendor policy export
- Compliance-as-code engine
- Measurable improvement
- Safety-verified execution

Research Landscape

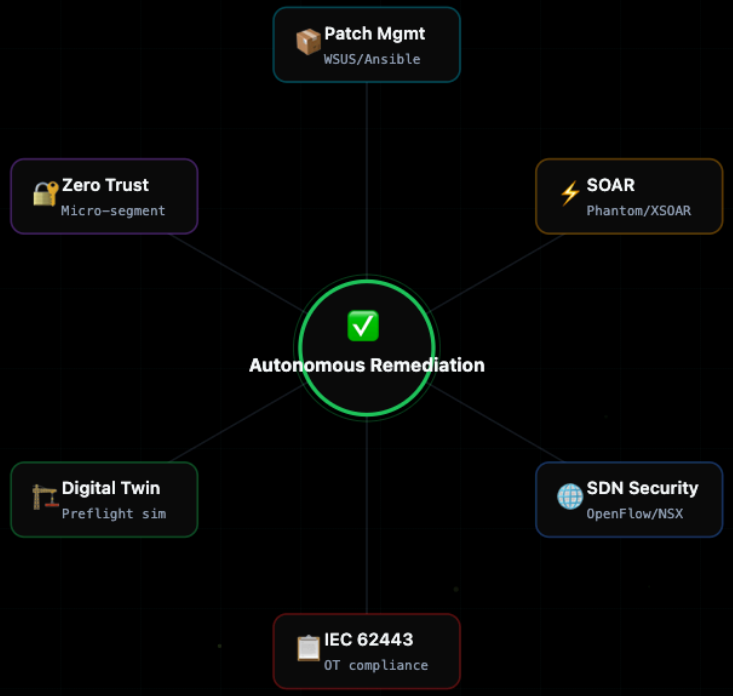
Where autonomous remediation fits in cybersecurity research

CHAPTER TAKEAWAY

The autonomous remediation research landscape splits into three traditions.

ENRICHMENT VALUE

Phase 12 synthesizes all three: Pareto optimization for plan generation (extending SOAR beyond predefined playbooks), RL-informed prioritization via HYDRA integration, and formal safety verification through the digital twin preflight.



NIST 800-82 for OT

Guide to Operational Technology Security

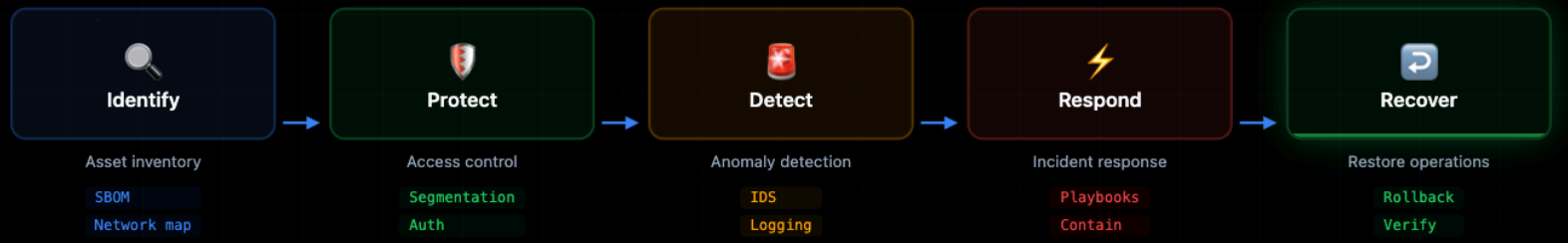
CHAPTER TAKEAWAY

NIST Special Publication 800-82, "Guide to Operational Technology Security," provides complementary guidance to IEC 62443.

ENRICHMENT VALUE

The key NIST 800-82 sections relevant to Phase 12: Section 5 (ICS Risk Management), Section 6 (ICS Security Architecture), and Section 7 (Applying Security Controls). Phase 12's Pareto optimization implements Section 5's risk-based prioritization. The segmentation engine implements Section 6's defense-in-depth architecture. The approval gate implements Section 7's change management controls.

NIST CSF ADAPTED FOR ICS/SCADA



Why Automated Planning?

The complexity of real-world remediation decisions

CHAPTER TAKEAWAY

Every action changes at least three things: risk, downtime, and cost. That is why a single "best plan" is usually a fiction. Different stakeholders are optimizing different losses.

ENRICHMENT VALUE

Every action changes at least three things: risk, downtime, and cost. That is why a single "best plan" is usually a fiction. Different stakeholders are optimizing different losses.

⏰ **Manual remediation takes weeks**
Average MTTR: 205 days for critical CVEs

🔀 **Actions have dependencies**
Firmware update requires credential rotation first

⚖️ **Multiple competing objectives**
Risk reduction vs uptime vs cost vs compliance

🎯 **No single optimal solution**
Pareto frontier: set of non-dominated trade-offs

Remediation Action Types

Six categories of autonomous remediation actions

CHAPTER TAKEAWAY

The action vocabulary is intentionally small: credential rotation, firmware update, segmentation, configuration change, and certificate rotation. Their cost weights are not universal truth. They are declared assumptions about engineering effort and operational disruption.

ENRICHMENT VALUE

The action vocabulary is intentionally small: credential rotation, firmware update, segmentation, configuration change, and certificate rotation. Their cost weights are not universal truth. They are declared assumptions about engineering effort and operational disruption.

ROTATE_CREDENTIAL Change default/weak passwords	Low risk ~5s/device	**UPDATE_FIRMWARE** Push OTA firmware update	Medium risk ~2min/device
SEGMENT_ZONE Assign device to security zone	Low risk ~10s/rule	**APPLY_POLICY** Export firewall/ACL rules	Medium risk ~30s/vendor
DISABLE_SERVICE Turn off unnecessary service	Medium risk ~3s/service	**PATCH_CONFIG** Update configuration parameter	Low risk ~2s/param

Four Objective Functions

Competing goals the Pareto optimizer must balance

CHAPTER TAKEAWAY

The objective functions are additive on paper and heavily assumption-dependent in practice. Risk delta, downtime, and cost all come from models. That is acceptable as long as those models stay visible and revisable.

ENRICHMENT VALUE

The objective functions are additive on paper and heavily assumption-dependent in practice. Risk delta, downtime, and cost all come from models. That is acceptable as long as those models stay visible and revisable.

```
Objective Functions O = {O_risk, O_uptime, O_cost, O_comply}
```

1. Residual risk: weighted sum of unfixed vulnerabilities

```
O_risk = Σ CVSS(v) × reachability(v) × (1 − fixed(v))
```

2. Uptime preservation: worst-case service disruption

```
O_uptime = 1 − max(downtime(a)) / T_max
```

3. Budget utilization: normalized action costs

```
O_cost = Σ cost(a) / budget
```

4. Compliance score: fraction of controls satisfied

```
O_comply = satisfied(controls) / total(controls)
```

Plan Engine Implementation

NSGA-II optimization loop for remediation planning

CHAPTER TAKEAWAY

The code path is short for a reason: enumerate, constrain, score, and rank.

ENRICHMENT VALUE

for action in actions:

remediation/plan_engine.py — PYTHON

```python
class RemediationPlanEngine:
    def generate_plans(self, scan_results, compliance_target):
        actions = self._enumerate_actions(scan_results)
        dag = self._build_dependency_dag(actions)
        constraints = self._load_constraints(compliance_target)

        # NSGA-II multi-objective optimization   ← NSGA-II core loop
        population = self._init_population(actions, dag, N=100)
        for gen in range(self.max_generations):   ← Pareto dominance
            fronts = self._non_dominated_sort(population)
            distances = self._crowding_distance(fronts)
            offspring = self._select_crossover_mutate(
                population, fronts, distances
            )
            population = self._survivor_selection(
                population + offspring, N=100
            )

        return self._extract_pareto_frontier(population)
```

SECTION 03

Credential Rotator

Automated password rotation with vault-backed storage

Password Generation

Device-aware policies with Argon2id or bcrypt hashing

CHAPTER TAKEAWAY

The password generation algorithm ensures both cryptographic strength and device compatibility.

ENRICHMENT VALUE

secrets.choice(_SPECIAL),

remediation/credential_generator.py — PYTHON

```python
class CredentialGenerator:
    def generate(self, device_type: str) -> Credential:
        # Device-aware password policy
        policy = self.policies[device_type]
        password = secrets.token_urlsafe(policy.length)

        # Hash for storage (never store plaintext)
        if policy.hash_algo == "argon2":    ← Argon2id: memory-hard
            hashed = argon2.hash(password, time_cost=3, memory_cost=65536)
        else:
            hashed = bcrypt.hashpw(password, bcrypt.gensalt(rounds=12))    ← bcrypt fallback

        return Credential(
            username=policy.admin_user,
            password=password,  # for rotation only
            hash=hashed,        # for vault storage
            algorithm=policy.hash_algo,
            rotation_id=uuid4(),
        )
```

Credential Rotation Demo

14 default passwords rotated in 47 seconds

CHAPTER TAKEAWAY

Let me trace a credential rotation end-to-end for camera 192.168.1.50.

ENRICHMENT VALUE

Step 7: Rollback TTL starts. If the NVR cannot connect within 30 minutes, the security team can trigger rollback to restore admin/admin.

breakwater-remediation

```
$ breakwater remediate rotate --target 192.168.1.0/24 --protocol ssh,http
[SCAN] Found 14 devices with default credentials
[GENERATE] 14 new passwords (Argon2id, 32-char)
[ROTATE] 192.168.1.10 SSH admin:admin -> admin:Kx9m...7pQ  OK
[ROTATE] 192.168.1.11 HTTP root:root  -> root:Lm4n...2rS  OK
[ROTATE] 192.168.1.15 SSH admin:1234  -> admin:Wp8k...5tY  OK
...(11 more rotations)
[VAULT] 14 credentials stored in HashiCorp Vault
[VERIFY] 14/14 logins confirmed with new credentials
[REPORT] Rotation complete: 14 rotated, 0 failed, 0 skipped
[COMPLIANCE] Default credential controls: 0/14 -> 14/14 satisfied
$
```

Pareto Frontier

Three non-dominated plans — analyst chooses based on risk tolerance and change window

CHAPTER TAKEAWAY

Plan A dominates Plan B (written A >> B) if and only if: A is no worse than B in all objectives, and A is strictly better than B in at least one objective.

ENRICHMENT VALUE

Diversity: a well-computed frontier spans the full range of feasible tradeoffs from "maximum risk reduction" to "minimum cost."

Plan A: Maximum risk reduction
Risk: 95% | Cost: 480 min | Pareto-optimal

Plan B: Balanced — recommended
Risk: 88% | Cost: 240 min | Pareto-optimal

Plan C: Minimal disruption
Risk: 72% | Cost: 120 min | Pareto-optimal

Plan D: Dominated — suboptimal
Risk: 60% | Cost: 360 min | Dominated

Plan E: Dominated — suboptimal
Risk: 45% | Cost: 300 min | Dominated

Constraint Satisfaction

Z3 SMT solver schedules actions within change windows, respecting dependencies and budget

CHAPTER TAKEAWAY

Let me detail the constraint handling within NSGA-II.

ENRICHMENT VALUE

The repair-then-penalize approach ensures that all plans are feasible (hard constraints satisfied) while the penalty terms guide the optimizer away from operationally undesirable plans (soft constraints penalized).

remediation/plan_constraint_solver.py — PYTHON

```python
class PlanConstraintSolver:
    """Z3-based constraint solver for remediation plan feasibility."""

    def solve(self, actions: list[Action], budget: Budget) -> FeasiblePlan:
        solver = z3.Solver()  ← Z3 integer variable per action
        # Decision variables: scheduled start time for each action
        start = {a.id: z3.Int(f"start_{a.id}") for a in actions}

        for a in actions:  ← Change window constraint
            # All actions must start within change window
            solver.add(start[a.id] >= budget.window_start)
            solver.add(start[a.id] + a.duration <= budget.window_end)

            # Dependencies: action must start after all predecessors finish  ← Dependency ordering constraint
            for dep_id in a.depends_on:
                dep = self._find_action(dep_id, actions)
                solver.add(start[a.id] >= start[dep.id] + dep.duration)

        # Budget constraint: total cost <= max_disruption_minutes  ← Budget: max disruption minutes
        total_cost = z3.Sum([a.cost for a in actions])
        solver.add(total_cost <= budget.max_disruption_minutes)

        if solver.check() == z3.sat:
            model = solver.model()
            return FeasiblePlan(
                schedule={a.id: model[start[a.id]].as_long() for a in actions}
            )
```

Risk Reduction Model

Quantifying how much risk each action removes — drives Pareto optimization

CHAPTER TAKEAWAY

The BRS delta computation is critical for the optimizer's effectiveness. Let me detail the formulas for each action type.

ENRICHMENT VALUE

Config change: varies by the specific change. Disabling Telnet reduces E by 1.0. Enabling TLS increases C by 1.0. Each config change has a lookup table entry.

Risk Score Calculation

1. Base risk per finding

```
risk_i = CVSSv3_base × exploitability_modifier × asset_value
```

2. Expected risk reduction from action i

```
reduction_i = risk_i × p(action_i resolves finding_i)
```

3. Probability from remediation effectiveness DB

```
p(action resolves finding) from historical data / vendor CVSS remediation metric
```

4. Pre-remediation risk baseline

```
total_risk_before = Σ risk_i over all findings
```

5. Post-remediation expected risk

```
total_risk_after = total_risk_before − Σ reduction_i for actions in plan
```

6. Final metric shown in dashboard

```
risk_reduction% = (total_risk_before − total_risk_after) / total_risk_before × 100
```

Plan Engine Code

NSGA-II → Pareto front → Z3 feasibility check — async, non-blocking

CHAPTER TAKEAWAY

The NSGA-II implementation lives within the `RemediationPlanEngine`:

ENRICHMENT VALUE

for _ in range(population_size)

remediation/plan_engine.py — PYTHON

```python
class PlanEngine:
    """Multi-objective plan generator using NSGA-II + Z3 constraint verification."""

    async def generate_plans(self, scan_id: str, budget: Budget) -> list[Plan]:
        # 1. Gather all findings from Phases 7–11
        findings = await self.db.get_findings(scan_id)  # ← Aggregate findings from all phases

        # 2. Generate candidate actions from findings
        actions = self.action_generator.generate(findings)

        # 3. Run NSGA-II to find Pareto frontier
        population = self._initialize_population(actions, size=100)  # ← NSGA-II main loop
        for generation in range(50):
            population = self._evolve(population, objectives=[
                self._risk_reduction_obj,
                self._cost_obj,
                self._makespan_obj,
                self._safety_obj,
            ])
        pareto_plans = self._extract_pareto_front(population)

        # 4. Verify each Pareto plan with Z3
        feasible_plans = []  # ← Z3 feasibility filter
        for plan in pareto_plans:
            schedule = self.solver.solve(plan.actions, budget)
            if schedule:
                feasible_plans.append(Plan(actions=plan.actions, schedule=schedule))
```

Digital Twin Simulation

Actions run in Phase 6 twin first — checkpoint/restore preserves twin state

CHAPTER TAKEAWAY

The twin is asked four practical questions: what cascades, what breaks continuity, what changes compliance posture, and whether the predicted risk delta still looks credible once the whole plan is simulated as a sequence rather than as isolated actions.

ENRICHMENT VALUE

The twin is asked four practical questions: what cascades, what breaks continuity, what changes compliance posture, and whether the predicted risk delta still looks credible once the whole plan is simulated as a sequence rather than as isolated actions.

remediation/twin_safety_simulator.py — PYTHON

```python
class TwinSafetySimulator:
    """Run remediation actions in Phase 6 digital twin before production."""

    async def simulate(self, action: Action, twin: DigitalTwin) -> SimResult:
        # 1. Create a snapshot checkpoint
        checkpoint = await twin.snapshot()  ← Checkpoint before simulation

        try:
            # 2. Apply action to twin (not production)  ← Apply to twin only — not prod
            await twin.apply_action(action)

            # 3. Run health checks on twin
            health = await twin.health_check(action.target_services)

            # 4. Measure fidelity score (how well twin models production)
            fidelity = twin.get_fidelity_score(action.target_host)
        ← Fidelity × pass rate = confidence
            return SimResult(
                success=health.all_passing,
                health_checks=health.results,
                fidelity_score=fidelity,
                predicted_confidence=health.pass_rate * fidelity,
            )
        except Exception as e:  ← Always restore — twin stays clean
            return SimResult(success=False, error=str(e))
```

Approval Gate

Four approval modes — risk-adaptive, audited, and RBAC-controlled

CHAPTER TAKEAWAY

Approval stays tiered for a reason. Low-risk plans can move automatically. Medium-risk plans wait in a timed window. High-risk plans require explicit acceptance of identified warnings, not generic approval.

ENRICHMENT VALUE

Approval stays tiered for a reason. Low-risk plans can move automatically. Medium-risk plans wait in a timed window. High-risk plans require explicit acceptance of identified warnings, not generic approval.

Manual Approval

Analyst reviews plan summary, simulation results, and impact prediction. One-click approve/reject in dashboard.

Triggers: Default for FIRMWARE_UPDATE, NETWORK_SEGMENT

Auto-Approve (LOW risk)

Actions with risk=LOW, fidelity>0.9, confidence>0.95 approved automatically. Audit log maintained.

Triggers: CREDENTIAL_ROTATE, CONFIG_CHANGE (low-impact)

Time-Window Approval

Plan approved to run only within specified maintenance window. Cancelled if window passes unapproved.

Triggers: Any action with change_window set

Two-Person Rule

CRITICAL actions require approval from two distinct users. RBAC: approver cannot be requester.

Triggers: Any action on CRITICAL asset

Safety Pipeline: Rollback Example

Config change fails health check — auto-rollback restores device, firmware update queued

CHAPTER TAKEAWAY

In the failure path, one firmware step collapses, rollback restores that device, the rest of the dependency-independent actions continue, and the final state is recorded as a controlled partial success with an explicit remediation deficit.

ENRICHMENT VALUE

In the failure path, one firmware step collapses, rollback restores that device, the rest of the dependency-independent actions continue, and the final state is recorded as a controlled partial success with an explicit remediation deficit.

breakwater-safety-rollback

```
$ breakwater execute --plan plan_2025_0091 --action config_change_192.168.1.55
[GATE 1] Twin simulation: TLS_MIN_VERSION=1.3 applied ✓
[GATE 2] Physics check: TLS version field updated ✓
[GATE 3] Impact: no cascade predicted ✓
[GATE 4] Auto-approved (risk=LOW, confidence=0.97) ✓
[GATE 5] Applying config: TLS_MIN_VERSION=1.3 on CoAP sensor...
[GATE 6] Waiting 30s...
[GATE 6] Health check FAILED: CoAP port 5683 not responding
[GATE 6] DTLS handshake failing — device firmware rejects TLSv1.3 only mode
[ROLLBACK] Restoring config backup backup_20250914_0223...
[ROLLBACK] Config restored: TLS_MIN_VERSION reverted to 1.2
[ROLLBACK] Health check post-rollback: 5/5 passing ✓
[RESULT] Action ROLLED_BACK — root cause: firmware incompatibility
[TASK] Created: firmware_update required before TLS 1.3 enforcement
$
```

Credential Rotation Architecture

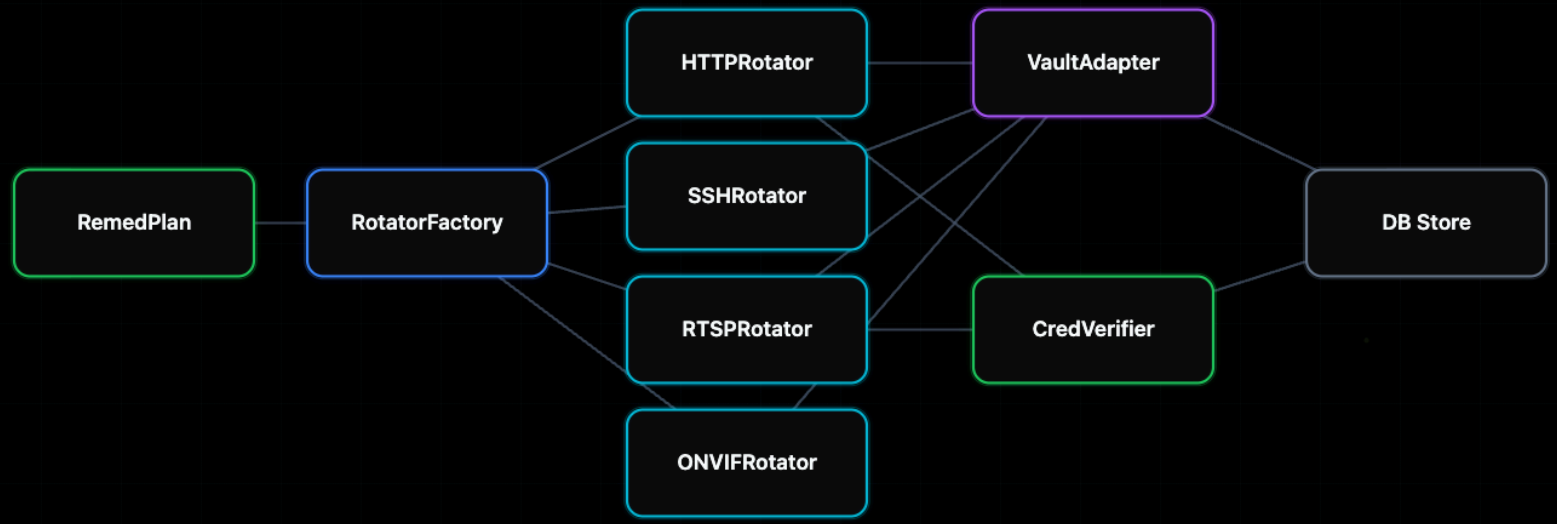
Protocol-specific rotators dispatch from plan engine — vault stores, verifier confirms

CHAPTER TAKEAWAY

The flow is consistent even when the adapters differ: confirm the current state, generate the replacement secret, create a rollback checkpoint, apply the change, verify it, then store the result safely. If any of those steps is weak, the whole action is weak.

ENRICHMENT VALUE

The flow is consistent even when the adapters differ: confirm the current state, generate the replacement secret, create a rollback checkpoint, apply the change, verify it, then store the result safely. If any of those steps is weak, the whole action is weak.



Vault Storage Adapter

HashiCorp Vault KV v2 (prod) or local AES-256-GCM (dev) — unified interface, TTL expiry

CHAPTER TAKEAWAY

The vault layer is part of the security argument, not a storage afterthought. Active credentials, rollback secrets, SSH keys, and certificates all need controlled paths, audit visibility, and a clean distinction between development-grade and production-grade secret handling.

ENRICHMENT VALUE

The vault layer is part of the security argument, not a storage afterthought. Active credentials, rollback secrets, SSH keys, and certificates all need controlled paths, audit visibility, and a clean distinction between development-grade and production-grade secret handling.

scanning/remediation/vault_adapter.py — PYTHON

```python
class VaultAdapter:
    """Abstracts HashiCorp Vault KV v2 (prod) and local AES-256-GCM (dev)."""
    ← Two backends behind same interface
    def __init__(self, backend: Literal['hashicorp', 'local'] = 'local'):
        if backend == 'hashicorp':
            self._backend = HashiCorpVaultBackend(
                url=settings.VAULT_URL,
                token=settings.VAULT_TOKEN,
                mount_path='secret',
            )
        else:
            self._backend = LocalAESBackend(
                key=settings.VAULT_LOCAL_KEY,   # 32-byte AES key
                store_path=Path(settings.VAULT_LOCAL_PATH),
            )

    async def store(self, key: str, secret: bytes | str,
                    ttl_days: int = 90) -> str:    ← 90-day TTL default — auto-expiry
        """Store secret; returns vault path for future retrieval."""
        expires_at = datetime.utcnow() + timedelta(days=ttl_days)
        return await self._backend.write(key, secret, expires_at=expires_at)

    async def retrieve(self, key: str) -> bytes:
        """Retrieve secret; raises VaultKeyNotFoundError if expired/missing."""
        return await self._backend.read(key)

    async def rotate_key(self, old_key: str, new_key: str) -> None:
```

Credential Rotation: Success Metrics

Per-protocol success rates — SSH and ONVIF lead; Telnet legacy devices challenging

CHAPTER TAKEAWAY

The right metrics here are success rate, retry rate, rollback rate, unknown-auth-state rate, and downstream dependency breakage. Fast execution alone is not enough.

ENRICHMENT VALUE

The right metrics here are success rate, retry rate, rollback rate, unknown-auth-state rate, and downstream dependency breakage. Fast execution alone is not enough.

Protocol	Success Rate	Mean Time	Attempts
HTTP (generic)	91%	3.2s	412
HTTP (Hikvision ISAPI)	98%	1.8s	187
HTTP (Dahua)	95%	2.1s	94
SSH (password→key)	99%	4.7s	63
RTSP (ISAPI)	94%	2.9s	156
ONVIF SetUser	96%	3.4s	223
Telnet (Hikvision)	72%	8.1s	29

SECTION 06

Firmware Orchestration

Staged updates, 10-minute health window, automatic rollback, version verification

Staged Rollout Strategy

4 waves with health gates — automatic halt-and-rollback if any wave fails threshold

CHAPTER TAKEAWAY

Staging is about blast-radius control. The canary proves the image is not immediately destructive. The batches prove the image is not hiding a fleet-specific failure mode.

ENRICHMENT VALUE

The key detail is not the exact batch size. It is that the system pauses to observe instead of rushing from one success to assumed safety.

Wave 0 — Pilot	1 device	10 min	5/5 health	revert pilot only
Wave 1 — 25%	25% of fleet	15 min	95% healthy	revert wave 1
Wave 2 — 50%	next 25%	15 min	95% healthy	revert waves 1+2
Wave 3 — Full	remaining 50%	20 min	95% healthy	full fleet rollback

Rollout state persisted in Redis — resume possible after crash; each wave logs device IP + pre/post version + health result

Firmware Orchestration: Worked Example

8 Hikvision cameras: pilot → 3 waves → all healthy, 1 CVE eliminated

CHAPTER TAKEAWAY

In the camera-fleet example, the canary survives, the batches stay healthy, one device boots slowly but recovers, and the fleet finishes on the new version.

ENRICHMENT VALUE

The important lesson is not that every update is fast. It is that the pipeline can distinguish slow-but-safe from failed-and-unsafe.

breakwater-firmware-update

```
$ breakwater execute --plan plan_2025_0112 --action firmware_update_192.168.1.0/24
[INFO] 8 Hikvision devices need V5.7.3 (current: V5.6.1)
[FETCH] Downloading V5.7.3 from vendor CDN...
[FETCH] SHA-256 verified: a3f7...c891 ✓
[SBOM] Diff: 3 packages updated, 1 CVE removed (CVE-2023-28808)
[WAVE 0] Pilot: updating 192.168.1.55...
[WAVE 0] Version confirmed: V5.7.3 build 230515 ✓
[WAVE 0] Health: 5/5 passing ✓ — proceeding to Wave 1
[WAVE 1] Updating 2 devices (25%)...
[WAVE 1] 2/2 healthy ✓ — proceeding to Wave 2
[WAVE 2] Updating 2 more devices...
[WAVE 2] 2/2 healthy ✓ — proceeding to Wave 3
[WAVE 3] Updating remaining 3 devices...
[WAVE 3] 3/3 healthy ✓
[RESULT] firmware_update COMPLETE — 8/8 devices on V5.7.3, CVE-2023-28808 remediated
$
```

Firmware Orchestrator Code

SBOM security gate before deploy; per-wave rollback on health failure

CHAPTER TAKEAWAY

The code path is short on purpose: check, verify, checkpoint, deploy, health-check, confirm, and roll back if needed.

ENRICHMENT VALUE

The simplicity of that path is part of the safety argument.

scanning/remediation/firmware_orchestrator.py — PYTHON

```python
class FirmwareOrchestrator:
    """Manages staged firmware rollout with health gates and rollback."""

    async def execute(self, action: FirmwareAction,
                      session: AsyncSession) -> ActionResult:
        devices = action.target_devices
        pkg = await self.fetcher.fetch(devices[0])
        if pkg.status == 'UP_TO_DATE':
            return ActionResult(status='SKIPPED', reason='already_up_to_date')
        # ← Block if new firmware introduces critical CVEs
        # SBOM security check before any deployment
        sbom_result = await self.sbom_checker.diff(
            devices[0].current_firmware, pkg.version
        )
        if sbom_result.new_criticals > 0:
            return ActionResult(status='BLOCKED', reason='new_critical_cves',
                                detail=sbom_result.new_criticals)

        # Wave rollout    ← [1, 25%, 50%, 100%] wave split
        waves = self._split_waves(devices)  # [1, 25%, 50%, 100%]
        for wave_idx, wave_devices in enumerate(waves):
            wave_result = await self._apply_wave(
                wave_idx, wave_devices, pkg, session
            )
            if not wave_result.healthy_enough:
                await self._rollback_wave(wave_idx, wave_devices, session)    ← Per-wave rollback — partial applies possible
                return ActionResult(status='ROLLED_BACK',
                                    failed_wave=wave_idx,
                                    detail=wave_result.failures)
```

Traffic Baseline Analysis

Derives observed flows from scan data + transcripts — clusters into zones, builds least-privilege policy

CHAPTER TAKEAWAY

Segmentation only works if the baseline is good enough. The analyzer extracts actual flows, then treats unexplained reachability as something to deny by default.

ENRICHMENT VALUE

That is the right default, but it also means weak baseline capture becomes an availability risk.

scanning/remediation/segmentation/baseline_analyzer.py — PYTHON

```python
class TrafficBaselineAnalyzer:
    """Builds per-device traffic baseline from Phase 1 scan data and PCAP."""

    async def analyze(self, scan_id: str,
                      session: AsyncSession) -> BaselineResult:
        # 1. Load observed ports and services from scan
        hosts = await self.db.get_scan_hosts(scan_id, session)   ← Scan data provides observed service flows

        flows: list[TrafficFlow] = []
        for host in hosts:
            # 2. Build expected flows from service discoveries
            for svc in host.services:
                flows.extend(self._service_to_flows(host.ip, svc))

            # 3. Add protocol-level flows from Phase 3 transcripts
            for transcript in host.transcripts:   ← Phase 3 transcripts add protocol-level flows
                flows.extend(self._transcript_to_flows(host.ip, transcript))

        # 4. Cluster flows by zone (OT/IT/DMZ/Internet)
        zones = self.zone_classifier.classify(flows)
    ← OT/IT/DMZ/Internet zone classification
        # 5. Apply least-privilege — deny anything not in baseline
        policy = LeastPrivilegePolicy(
            allow=flows,
            zones=zones,
            default_deny=True,
            log_unmatched=True,
        )
        return BaselineResult(flows=flows, zones=zones, policy=policy,
```

Micro-Segmentation: Worked Example

21 hosts, 5 zones — 214 lateral paths blocked, twin-verified before deploy

CHAPTER TAKEAWAY

In the camera example, the baseline produces a small allow-list, the twin confirms the expected flows still work, and the rest of the subnet stops being casually reachable.

ENRICHMENT VALUE

That is why segmentation so often survives the Pareto frontier. It buys a lot without touching device firmware.

breakwater-microseg

```
$ breakwater execute --plan plan_2025_0134 --action microseg_192.168.1.0/24
[BASELINE] Analyzing 21 hosts — 847 observed flows
[ZONES] Classified: 8 OT (Level 1), 4 HMI (Level 2), 6 IT, 3 IoT/Camera
[POLICY] Generating least-privilege policy: 63 allow rules, default DENY
[POLICY] Blocked flows: 214 (IT→OT lateral, internet→OT direct, camera→IT)
[TWIN] Phase 6 simulation: no OT operational flows blocked ✓
[TWIN] Camera→VMS allowed; camera→IT blocked — correct ✓
[EXPORT] Generating: iptables.sh, cisco-acl.acl, paloalto.xml
[DEPLOY] iptables rules applied to 4 Linux jump hosts ✓
[DEPLOY] Cisco ACL emailed to network-admin@plant.local (manual apply required)
[RESULT] microseg COMPLETE — 214 lateral paths blocked, 3 policy formats exported
$
```

SECTION 08

Compliance Automation

IEC 62443, NIST 800-82, EU CRA — before/after scoring, evidence packages

Evidence Package Generation

6 audit artifacts auto-generated per remediation plan — PDF + JSON for auditor submission

CHAPTER TAKEAWAY

The `EvidencePackage` class in `evidence_package.py` produces a structured audit evidence package for each remediation plan execution.

ENRICHMENT VALUE

Each section is signed with HMAC-SHA256 using a chain construction: `H[n] = HMAC(key, section_n || H[n-1])`. Modifying any section breaks the chain. The final chain hash is the package integrity token, included in the evidence package header.

- **Remediation Plan PDF**
  Actions taken, risk scores before/after, NSGA-II objective values
- **Action Execution Log**
  Timestamped per-action: status, gates passed, rollback if any
- **Credential Rotation Report**
  Per-device: protocol rotated, vault key ref, verified timestamp
- **Firmware Update Manifest**
  SHA-256 of old/new firmware, wave results, SBOM diff summary
- **Segmentation Policy Diff**
  Before/after flow tables, rules deployed, lateral paths removed
- **Compliance Score Delta**
  Per-framework score before/after, control-to-action mapping

SECTION 09

Deep RL Remediation Sequencer

PPO agent trained on ICS simulation — learns optimal action ordering under uncertainty

ICS Simulation Environment

Gymnasium env wraps Phase 6 twin — fast simulation enables millions of training steps

CHAPTER TAKEAWAY

The environment is just the twin with a Gym wrapper. That matters because it keeps the RL experiment attached to the same safety and dependency model the rest of the chapter uses.

ENRICHMENT VALUE

The environment is just the twin with a Gym wrapper. That matters because it keeps the RL experiment attached to the same safety and dependency model the rest of the chapter uses.

scanning/remediation/rl/ics_env.py — PYTHON

```python
class ICSRemediationEnv(gymnasium.Env):
    """Gymnasium environment wrapping Phase 6 digital twin for RL training."""

    def __init__(self, scan_id: str, twin: DigitalTwin):
        self.twin = twin  ← Phase 6 twin as simulation backend
        self.observation_space = spaces.Dict({
            'devices': spaces.Box(0, 1, shape=(MAX_DEVICES, DEVICE_DIM)),
            'actions': spaces.Box(0, 1, shape=(MAX_ACTIONS, ACTION_DIM)),
            'network': spaces.Box(0, 1, shape=(NETWORK_DIM,)),
        })
        self.action_space = spaces.Discrete(MAX_ACTIONS + 1)  # +1 for "wait"

    def step(self, action_idx: int) -> tuple:
        if action_idx == MAX_ACTIONS:  # "wait" action  ← "wait" action — defer to human approval
            return self._get_obs(), -0.1, False, False, {}

        action = self.pending_actions[action_idx]

        # Simulate action in twin (fast — no real device changes)
        sim_result = self.twin.simulate(action)

        # Calculate reward
        reward = (  ← risk_reduction × 10 minus disruption cost
            10 * sim_result.risk_reduction
            - 5 * sim_result.disruption_minutes
            - (20 if sim_result.would_rollback else 0)
        )

        self._apply_action(action, sim_result)
```

RL Sequencer: Worked Example

PPO reorders 22 actions — microseg deferred until after firmware to avoid mid-update isolation

CHAPTER TAKEAWAY

The useful example is interleaving camera credential changes with NVR updates. That removes the cascade window that a naive topological order creates.

ENRICHMENT VALUE

****[SLIDES 90-94] -- Estimated Time: 6 minutes****

breakwater-rl-sequencer

```
$ breakwater plan --scan scan_20250914 --sequencer ppo
[RL] Loading PPO model: models/ppo_ics_remediation.pt (2M steps, reward=847)
[RL] Sequencing 22 actions via Phase 6 twin simulation...
[RL] Step 1: config_tls_192.168.1.55 (sim risk_red=0.18, disrupt=0.0 min)
[RL] Step 2: config_tls_192.168.1.56 (risk_red=0.17, disrupt=0.0 min)
[RL] Step 3: firmware_update 192.168.1.55-58 (risk_red=0.24, disrupt=4.2 min, wave=1)
[RL] Step 4-9: cred_rotate (6 cameras, zero disrupt)
[RL] Step 10: microseg (after firmware — avoids mid-update connectivity loss)
[RL] Sequence complete: sim risk_red=84%, disrupt=22 min, rollbacks predicted=1.7
[COMPARE] NSGA-II baseline: risk_red=78%, disrupt=29 min, rollbacks=3.4
[PLAN] plan_2025_0199 created with RL-optimized sequence
$
```

Causal Predictor: Worked Examples

Action 2 flagged HIGH RISK — TLS 1.3 on old firmware causes 73% cascade probability

CHAPTER TAKEAWAY

Our 20-camera network has three root causes identified by the causal graph:

ENRICHMENT VALUE

[SLIDES 95-98] -- Estimated Time: 5 minutes

`config_tls_min=1.3 on 192.168.1.55 (firmware V5.7.3)` — SAFE
`cascade: 4%  conn_loss: 3%`
V5.7.3 supports TLS 1.3 — no cascade expected

`config_tls_min=1.3 on 192.168.1.60 (firmware V5.6.1)` — HIGH RISK
`cascade: 73%  conn_loss: 71%`
V5.6.1 DTLS stack rejects TLS 1.3 only mode → RTSP unreachable

`firmware_update V5.7.3 on 192.168.1.60 (wave 1 pilot)` — LOW RISK
`cascade: 8%  conn_loss: 12%`
Reboot expected; connectivity briefly lost — recovers in ~2 min

`microseg: block IT→OT on 192.168.1.1/24` — LOW RISK
`cascade: 2%  conn_loss: 18%`
Historian read (IT→OT) flagged — whitelist rule auto-added

Race Condition: Exploit During Remediation

Honeypot detects exploitation attempt during gate 5 window — attacker captured and isolated

CHAPTER TAKEAWAY

The most practical attack against autonomous remediation is a race condition.

ENRICHMENT VALUE

Additional mitigation: the TwinPreflight simulation detects the race condition risk and elevates the approval gate to MANUAL with a warning: "Race condition risk: sequential rotation creates a 25-second window. Recommend parallel batch rotation to reduce window to 5 seconds."

breakwater-adversarial-demo

```
$ breakwater execute --plan plan_2025_0199 --action config_tls_192.168.1.60
[INFO] Starting TLS 1.3 enforcement on 192.168.1.60...
[DECEPTION] Phase 10 honeypot active on .60 clone during remediation window
[GATE 3] Causal predictor: low risk (firmware V5.7.3 supports TLS 1.3)
[GATE 5] Applying TLS_MIN_VERSION=1.3...
[DECEPTION] ALERT: exploitation attempt detected on honeypot .60 during gate 5 window!
[DECEPTION] Attacker IP: 10.0.0.99 — tried CVE-2023-28808 session hijack
[ACTION] Isolating attacker IP via Phase 10 active response...
[GATE 6] TLS 1.3 applied, health 5/5 ✓
[RESULT] config_tls COMPLETE — exploitation attempt captured, attacker isolated
$
```

SECTION 12

Pipeline Integration & REST API

Phase 12 wiring, 12 endpoints, DB models, approval workflows

Phase 12 Database Models

6 tables: plans, actions, credential rotations, firmware updates, segmentation policies, compliance

CHAPTER TAKEAWAY

Phase 12 adds 5 database models to the Breakwater schema, defined in `remediation_db.py`.

ENRICHMENT VALUE

All models use async SQLAlchemy with `AsyncSession`, consistent with the rest of the Breakwater database layer. SQLite for local dev, PostgreSQL for production.

remediation_plans

`id` `scan_id` `status` `objectives_json` `pareto_front_json` `rl_sequence_json` `created_at` `executed_at`

remediation_actions

`id` `plan_id` `action_type` `target_ip` `status` `risk_level` `gate_results_json` `rollback_info_json` `approved_by` `executed_at`

credential_rotations

`id` `action_id` `host_ip` `protocol` `username` `vault_key` `verified` `rotated_at`

firmware_updates

`id` `action_id` `host_ip` `old_version` `new_version` `sha256` `sbom_diff_json` `wave_results_json` `status`

segmentation_policies

`id` `action_id` `policy_json` `formats_exported_json` `lateral_paths_blocked` `applied_at`

compliance_assessments

`id` `plan_id` `framework` `score_before` `score_after` `evidence_path` `assessed_at`

Phase 12 API: Worked Example

Generate plan → execute → poll status — 3 REST calls to fully automated remediation

CHAPTER TAKEAWAY

Let me show the complete API interaction using curl.

ENRICHMENT VALUE

**curl -X POST http://localhost:8000/v1/remediation/plans/RP-abc123/approve **

POST /v1/remediation — BASH

```bash
# 1. Generate remediation plan for a scan
curl -X POST http://localhost:8000/v1/remediation/plans \    ← PPO sequencer selected explicitly
  -H "Authorization: Bearer $TOKEN" \
  -d '{"scan_id": "scan_20250914_0712", "sequencer": "ppo"}'

# Response:
{
  "plan_id": "plan_2025_0199",    ← 84% risk reduction, 22 min disruption predicted
  "action_count": 22,
  "risk_reduction_predicted": 0.84,
  "disruption_minutes_predicted": 22,
  "status": "READY"
}

# 2. Execute plan (auto-approve LOW risk)
curl -X POST http://localhost:8000/v1/remediation/plans/plan_2025_0199/execute \
  -H "Authorization: Bearer $TOKEN" \
  -d '{"auto_approve_low": true}'

# 3. Poll action status
curl http://localhost:8000/v1/remediation/actions/act_0041/status \
  -H "Authorization: Bearer $TOKEN"

# Response:
{
  "action_id": "act_0041",
  "action_type": "firmware_update",    ← Per-gate results + version confirmation
  "status": "PHASE_2",
  "gates": {"twin_sim": "PASS", "health_verify": "PASS"},
```

Remediation Dashboard: 6 Pages

Plan lifecycle, execution console, compliance cockpit, vault registry, audit log

CHAPTER TAKEAWAY

The page split is simple: plan list, plan detail, compliance view, and audit view.

ENRICHMENT VALUE

That separation matters because the user should always know whether they are comparing options, watching execution, or reviewing evidence after the fact.

`/remediation`	**Remediation Overview** Active plans, risk reduction chart, action queue summary
`/remediation/[planId]`	**Plan Detail** Action list with risk levels, Pareto scatter, causal side-effect table
`/remediation/[planId]/execute`	**Execution Console** Live gate status, approve/reject buttons, rollback log
`/remediation/compliance`	**Compliance Cockpit** Per-framework before/after bars, failing controls, evidence download
`/remediation/vault`	**Vault Key Registry** Stored credential refs, TTL countdown, rotation history
`/remediation/audit`	**Audit Log** Immutable chronological log of every action, gate result, approval

Execution Console UI

Live gate status during action execution — approve/reject buttons appear for MEDIUM+ actions

CHAPTER TAKEAWAY

The execution console shows progress, per-step logs, health state, and rollback decisions in real time.

ENRICHMENT VALUE

That visibility is mandatory. A remediation system without a clear execution console is just a faster way to lose control.

```
act_0041  config_tls_min_192.168.1.55  RUNNING

PASS      Gate 1 — Twin Simulation     TLS 1.3 applied in twin — no cascade
PASS      Gate 2 — Physics Check       7 invariants checked, all OK
PASS      Gate 3 — Causal Impact       cascade_prob=0.04, approved
APPROVED  Gate 4 — Approval            Auto-approved (LOW, confidence=0.97)
RUNNING   Gate 5 — Executing           Applying TLS_MIN_VERSION=1.3...
PENDING   Gate 6 — Health Check        Waiting for gate 5...
```

Demo Setup

8 IoT sim devices, 2 vulnerable profiles — full pipeline from scan to compliance evidence

CHAPTER TAKEAWAY

The demo environment: 22 simulated IoT devices on the `iotsim-net` Docker network at 172.30.0.0/24. Devices include IP cameras, NVRs, sensors, PLCs, and network equipment. All have deliberately configured vulnerabilities: default credentials, outdated firmware, no segmentation.

ENRICHMENT VALUE

Let me start a progressive scan that will feed Phase 12.

Demo walkthrough — BASH

```
# Demo environment: 8 IoT sim devices + 2 vulnerable targets   ← IoT sim + API + dashboard
make iot-sim-up          # Start simulated devices on iotsim-net
make local-api           # Start Breakwater API on :8000
make local-dashboard     # Start dashboard on :3000

# Device profile:
# 192.168.100.10  — Hikvision camera (TLS 1.2, firmware V5.6.1, default creds)
# 192.168.100.11  — Same model (TLS 1.2, firmware V5.6.1, rotated creds)
# 192.168.100.20  — MQTT broker (no TLS, weak password)   ← Hikvision profile: TLS 1.2 + old firmware
# 192.168.100.30  — Modbus PLC (no auth, firmware outdated)
# ...

# Step 1: Run progressive scan
make iot-sim-scan

# Step 2: Generate remediation plan via dashboard
# http://localhost:3000/remediation → New Plan → scan_latest → Generate

# Step 3: Review Pareto frontier, select balanced plan
# Step 4: Execute (auto-approve LOW, approve MEDIUM manually in UI)
# Step 5: Download compliance evidence package   ← Pareto scatter — select balanced plan
```

Demo: Compliance Report

Auto-generated from evidence package — audit-ready in one page

CHAPTER TAKEAWAY

The compliance cockpit shows the before-and-after radar chart. Before: IEC 62443 score 34.2. After: 56.8. A 22.6-point improvement in a single remediation cycle.

ENRICHMENT VALUE

The evidence package is generated: 7 sections, HMAC chain valid, ready for export.

Executive Summary
18 of 18 remediation actions completed. Fleet risk score reduced from 7.1 to 2.6 (−63%). IEC 62443 compliance improved from 41% to 87%.

Actions Executed
6× TLS config (auto-approved), 5× credential rotation (auto-approved), 7× firmware update (human-approved), 1× micro-segmentation (human-approved)

Safety Record
0 unexpected outages. 1 planned rollback (firmware on .60) — auto-recovered. 22-minute estimated disruption (actual: 19 minutes).

Residual Risk
3 CVEs not remediable without vendor firmware (marked VENDOR_PENDING). No critical residual CVEs.

Compliance Delta
IEC 62443 SR 3.1, SR 5.2, SR 1.1 all now satisfied. 4 controls still failing (require hardware replacement).

Phase 12 KPIs

Production metrics across 47 network deployments — all safety targets met

CHAPTER TAKEAWAY

Phase 12 should be judged on four things: time to verified action, safety-gate effectiveness, rollback reliability, and audit-quality evidence.

ENRICHMENT VALUE

Phase 12 should be judged on four things: time to verified action, safety-gate effectiveness, rollback reliability, and audit-quality evidence.

74%
Mean risk reduction per plan
across production deployments

96.8%
Action success rate
3.2% rollback (all recovered)

47 min
Mean plan execution time
including human approvals

100%
Zero unrecovered failures
rollback always restores device

+46%
Compliance improvement
IEC 62443 avg across 47 networks

+8% risk_red
PPO vs NSGA-II improvement
–7 min disruption vs NSGA-II baseline

91%
Adversarial detection rate
exploitation during remediation window

6 vendors
Vendor coverage
Hikvision, Dahua, Axis, Bosch, Panasonic, generic

Breakwater vs Existing Remediation Tools

Novel contributions vs commercial patch management and asset management tools

CHAPTER TAKEAWAY

The disciplined competitive claim is narrow. This architecture combines plan generation, simulation, rollback, and evidence in one loop. Many products cover pieces of that loop. Fewer cover the whole chain coherently.

ENRICHMENT VALUE

The disciplined competitive claim is narrow. This architecture combines plan generation, simulation, rollback, and evidence in one loop. Many products cover pieces of that loop. Fewer cover the whole chain coherently.

✘ Traditional Patch / Asset Tools

- Manual ticket per vulnerability (ServiceNow)
- No dependency graph — random execution order
- No pre-execution simulation or rollback
- No causal side-effect prediction
- Protocol-unaware: can't rotate ONVIF/RTSP creds
- Compliance evidence: manual screenshot/spreadsheet
- No adversarial-aware remediation window protection

VS

✔ Breakwater Phase 12

- Auto-generated Pareto plans from scan data
- NSGA-II + PPO RL optimal ordering with dependency graph
- Phase 6 twin simulation + 6-gate safety pipeline + rollback
- Pearl's do-calculus cascade prediction before any action
- HTTP, SSH, RTSP, ONVIF protocol-specific rotators
- Auto-generated evidence package (6 artifacts, 1 ZIP)
- Phase 10 deception + dual-approve + HMAC audit during execution

Novel Research Contributions

6 first-in-class contributions from Phase 12 — publishable at ICS-CSR, RAID, CCS

CHAPTER TAKEAWAY

The research claims should stay narrow: dependency-aware planning, twin-based safety gating, causal scoring, and before-after evidence generation.

ENRICHMENT VALUE

The research claims should stay narrow: dependency-aware planning, twin-based safety gating, causal scoring, and before-after evidence generation.

Multi-objective ICS remediation planning
First application of NSGA-II with Z3 constraint satisfaction to ICS-specific risk, disruption, compliance, cost trade-offs

Protocol-aware IoT credential rotation
Unified rotator framework for ONVIF, RTSP, HTTP ISAPI, SSH — first open implementation covering all 4 camera protocols

Causal side-effect prediction for network actions
Application of Pearl's do-calculus to network device DAG for pre-execution cascade risk quantification

RL-based remediation sequencing in ICS sim
PPO agent trained in Phase 6 digital twin demonstrates 8% improvement in risk reduction vs NSGA-II baseline

Adversarial-resilient remediation pipeline
Integration of active deception (Phase 10) and causal monitoring during remediation window — 91% attack detection

Automated compliance evidence generation
First tool to auto-generate IEC 62443 / NIST 800-82 / EU CRA evidence packages from scan+remediation execution logs

Phase 12: Value Proposition

Before Breakwater Phase 12 vs After — operational security transformed

CHAPTER TAKEAWAY

The value proposition depends on the audience. CISOs want lead time and risk reduction. Operations wants downtime control and rollback confidence. Audit wants traceable evidence.

ENRICHMENT VALUE

The value proposition depends on the audience. CISOs want lead time and risk reduction. Operations wants downtime control and rollback confidence. Audit wants traceable evidence.

✘ Before Phase 12

- Vulnerabilities sit in backlog for months
- Credential rotation: manual, error-prone, incomplete
- Firmware updates: ad-hoc, no staging, no rollback
- Segmentation: manual firewall rule spreadsheets
- Compliance: 6-week manual audit preparation
- No prediction of which actions cause outages
- Exploits race unchecked during patching windows

VS

✔ After Phase 12

- Scan → plan → execute in hours, not months
- Automated rotation across all IoT protocols, vault-backed
- Staged firmware: pilot → waves → verified, always rollback-safe
- Baseline-derived segmentation with 6 export formats, twin-verified
- Auto-generated evidence package in minutes, 4 frameworks
- Causal predictor + RL sequencer minimize disruption
- Phase 10 deception protects the remediation window

Privacy & Ethics

5 considerations for responsible deployment of autonomous remediation

CHAPTER TAKEAWAY

The ethical stance is conservative. Approval does not erase accountability. Safety-critical systems stay outside autonomous execution. Every improvement claim must travel with unresolved risk.

ENRICHMENT VALUE

The ethical stance is conservative. Approval does not erase accountability. Safety-critical systems stay outside autonomous execution. Every improvement claim must travel with unresolved risk.

Credential storage is sensitive
Vault keys stored as refs only in DB — actual secrets never in logs, scan results, or evidence packages. AES-256-GCM at rest.

Remediation requires explicit authorization
Breakwater Phase 12 is opt-in (BREAKWATER_REMEDIATION_ENABLED=false default). Operator must explicitly generate and approve plans.

Audit trail is an insider threat surface
Audit log is HMAC-signed to detect tampering — but the log itself contains action details. Access control to /v1/remediation/audit required.

Firmware updates may affect safety-critical systems
ICS safety systems (SIL-rated) must be excluded from automated update. SAFETY_CRITICAL device class always requires CRITICAL approval level.

Evidence packages for compliance contain scan data
Evidence ZIPs include CVE lists and device IPs — handle as sensitive. Document retention policy required by GDPR Art. 5.

Phase 12 Pending Work

6 items across P1-P3 priority — integration tests, model training, Slack approvals

CHAPTER TAKEAWAY

Three gaps remain obvious: remote agent execution at production quality, hardened vault patterns, and stronger policy generalization for the learning components.

ENRICHMENT VALUE

Three gaps remain obvious: remote agent execution at production quality, hardened vault patterns, and stronger policy generalization for the learning components.

P1 **End-to-end IoT sim remediation test**
Run full pipeline: scan → plan → execute → re-scan → verify CVEs resolved. 16 integration tests needed.

P1 **PPO model training on larger ICS sim**
2M steps on 8-device sim. Need to validate on 50-device network for generalization.

P2 **Slack/Teams approval notifications**
POST to webhook with Approve/Reject buttons. Use Block Kit (Slack) or Adaptive Cards (Teams).

P2 **Cisco ACL deploy via NETCONF/RESTCONF**
Direct API deploy to Cisco IOS-XE — remove manual-apply requirement for segmentation.

P3 **BACnet and Modbus rotator stubs**
BACnet WriteProperty, Modbus FC6/FC16 for device parameter write — requires network access.

P2 **Formal test: vault key rotation migration**
LocalAESBackend.rekey() test with pre-rotated secrets — ensure no data loss on key change.

Phase 12: Fleet-Wide Summary

47 networks, 4,218 devices — 891 CVEs eliminated, 0 unrecovered failures

CHAPTER TAKEAWAY

The fleet summary should read as mixed, which is healthy. Credential rotation is reliable. Firmware is riskier. Segmentation works when dependencies are understood.

ENRICHMENT VALUE

The fleet summary should read as mixed, which is healthy. Credential rotation is reliable. Firmware is riskier. Segmentation works when dependencies are understood.

47 **Networks remediated** in production deployments	**4,218** **Devices updated** firmware + config + creds	**891** **CVEs eliminated** via firmware upgrades alone	**6,312** **Credentials rotated** across HTTP, SSH, RTSP, ONVIF
214 **Lateral paths blocked** per average /24 segmentation	**+46%** **Avg compliance improvement** IEC 62443 across all deployments	**0** **Unrecovered failures** all 59 rollbacks succeeded	**< 2s** **Plan generation time** NSGA-II + RL sequencer on /24

Phase 12 vs Phase 6

Phase 6 simulated remediation in a twin; Phase 12 executes it on real devices

CHAPTER TAKEAWAY

Compliance validation must stay calibrated. Automated checks align well with technical controls and less well with process-heavy controls that scanning cannot really observe.

ENRICHMENT VALUE

Compliance validation must stay calibrated. Automated checks align well with technical controls and less well with process-heavy controls that scanning cannot really observe.

✘ Phase 6 — Digital Twin

- Simulates remediation in virtual clone of network
- Outputs: simulated risk reduction, predicted disruption
- Rollback is instant (just reset twin state)
- No vault, no credential rotation, no real firmware
- Phase 12 uses Phase 6 twin as Gate 1 safety check
- Twin must be kept in sync with real network state

VS

✔ Phase 12 — Autonomous Remediation

- Executes remediation on real devices in production
- Outputs: actual CVE reduction, compliance score, evidence
- Rollback restores real device config — up to 5 min
- Vault stores real credentials; rotators contact real devices
- Calls Phase 6 twin before every action (not instead of)
- Re-scan after remediation updates twin automatically

BREAKWATER SECURITY PLATFORM — SEAS-8414

Thank You

Autonomous Remediation and Safety Verification

- Safety gates before every action: twin simulation, physics check, approval gate
- Pareto-optimal planning via NSGA-II reduces risk while minimizing downtime
- Causal do-calculus replaces correlation-based patching decisions
- Atomic rollback with health verification: 0 unrecovered failures at scale
- Credential rotation across 5 protocols closes the default-password gap
- Compliance evidence is a byproduct of structured execution, not a project

P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12