SEAS-8414 Week 12 Student Lab Guide

SEAS-8414 Week 12 Student Lab Guide: Autonomous Remediation

Start Here: Download the Student ZIP

Before running any lab commands, download the Week 12 student package from the course site:

https://8414.bwater.io/downloads/labs/packages/seas8414-blackboard-week-12-2026.05.0_4e76a52f_aws8414.zip

The recommended path is the package Makefile:

```
unzip seas8414-blackboard-week-12-2026.05.0_4e76a52f_aws8414.zip
cd seas8414-blackboard-week-12-2026.05.0_4e76a52f_aws8414
make week12
```

The Makefile calls `run-week12-lab.sh`, extracts the nested runtime ZIP, starts the lab, runs the Week 12 API workflow, saves evidence under `lab-results/week-12/evidence/`, generates `lab-results/week-12/index.html`, and cleans up containers at exit.

You can also download the runner directly from https://8414.bwater.io/downloads/labs/scripts/run-week12-lab.sh.

Manual extraction uses two ZIP layers. First extract the weekly Blackboard ZIP, then extract the nested runtime ZIP:

```
unzip seas8414-blackboard-week-12-2026.05.0_4e76a52f_aws8414.zip
cd seas8414-blackboard-week-12-2026.05.0_4e76a52f_aws8414
unzip runtime/seas8414-student-lab-2026.05.0+4e76a52f.aws8414.zip
cd seas8414-student-lab-2026.05.0+4e76a52f.aws8414/student-lab
```

Run the instructions in this guide from that `student-lab/` directory. The matching screencast and LLM prompt are published next to the ZIP on the labs page:

- Screencast MP4: https://8414.bwater.io/downloads/labs/screencasts/phase12-lab-screencast.mp4
- LLM Prompt: https://8414.bwater.io/downloads/labs/prompts/phase12-lab-llm-prompt.md
- Run Script: https://8414.bwater.io/downloads/labs/scripts/run-week12-lab.sh

---

Breakwater Phase 12: Autonomous Remediation Lab

Phase 12 introduces closed-loop autonomous remediation: AI-optimized remediation plan generation using Pareto multi-objective optimization, preflight simulation in the Phase 6 digital twin before any production change, safety-gated human approval workflow, automated credential rotation, staged firmware update orchestration, segmentation policy generation (iptables/nftables format), IEC 62443 compliance scoring before and after,

and cryptographic audit evidence packaging. These capabilities complete the Breakwater security lifecycle: from discovery (Phase 1) to remediation (Phase 12), with every action traceable, reversible, and verifiable.

(See Slides 001-009 for Phase 12 overview, remediation architecture, and safety pipeline)

Prerequisites

- Completed Phase 1 lab (scan data with vulnerabilities and default credentials)
- Completed Phase 4 lab (attack graph and BRS scores for remediation prioritization)
- Completed Phase 6 lab (digital twin available for preflight simulation)
- A completed scan with `$SCAN_ID` and `$TOKEN` variables set (see Phase 1 Exercise 6)
- Understanding of network policy concepts (firewall rules, VLANs)

If you need to set up your variables from a previous session:

```
# Login and capture token
TOKEN=$(curl -s -X POST http://localhost:8100/v1/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"student@example.com","password":"SecurePass!2026"}' \
  | jq -r '.access_token')

# Get the most recent completed scan ID
SCAN_ID=$(curl -s "http://localhost:8100/v1/scanning/smart-
        scan/history?limit=1" \
  -H "Authorization: Bearer $TOKEN" \
  | jq -r '.scans[0].scan_id')

echo "Token: ${TOKEN:0:20}..."
echo "Scan ID: $SCAN_ID"
```

What you should see:

```
Token: eyJhbGciOiJIUzI1Ni...
Scan ID: a3f1c2d4-5e6f-7890-abcd-ef1234567890
```

> **Troubleshooting:** If `Token: null` appears, verify the user account exists. If `Scan ID: null`, run `python scan_report.py 172.30.0.0/24` to create one.

Phase 12 API Cheatsheet

Endpoint	Method	Descrip
`/v1/remediation/{scan_id}/plan`	POST	Generate remediati plan (Par optimize
`/v1/remediation/{scan_id}/plan/{plan_id}`	GET	Retrieve specific p
`/v1/remediation/{scan_id}/plan/{plan_id}/simulate`	POST	Preflight simulatio digital tw
`/v1/remediation/{scan_id}/plan/{plan_id}/approve`	POST	Human approval (required before executio
`/v1/remediation/{scan_id}/plan/{plan_id}/execute`	POST	Execute approved plan step step

Endpoint	Method	Descrip
`/v1/remediation/{scan_id}/credentials`	POST	Autonom credentia rotation
`/v1/remediation/{scan_id}/firmware`	POST	Staged firmware update orchestra
`/v1/remediation/{scan_id}/segmentation`	GET	Generate network segmenta policy
`/v1/remediation/{scan_id}/compliance/before`	GET	IEC 624 complian score (pr remediati
`/v1/remediation/{scan_id}/compliance/after/{plan_id}`	GET	IEC 624 complian score (po remediati
`/v1/remediation/{scan_id}/audit`	GET	Export cryptogra audit evidence package

All Phase 12 endpoints require Bearer token authentication.

(See Slides 081-090 for API design, endpoint reference, and audit architecture)

Exercise 1: Generate Remediation Plan

(See Slides 010-025 for Pareto optimization, multi-objective planning, and do-calculus counterfactuals)

Breakwater generates remediation plans using Pareto multi-objective optimization: the plan simultaneously minimizes risk (maximize BRS reduction), minimizes operational disruption (minimize downtime), and minimizes cost (minimize engineering hours). The Pareto-optimal plan is the set of actions where no single objective can be improved without worsening another. Students select from the Pareto frontier based on their organization's priority weighting.

Architecture: What This Exercise Tests

```
graph LR
    subgraph "Risk Input (Phase 4)"
        A[BRS Scores] --> B[ParetoOptimizer]
        C[Attack Paths] --> B
        D[CVE Assessment] --> B
    end
    subgraph "Pareto Optimization (remediation/)"
        B --> E[Objective1: MaxBRSReduction]
        B --> F[Objective2: MinDowntime]
        B --> G[Objective3: MinCost]
        E & F & G --> H[ParetoFrontier]
    end
    subgraph "API Layer (remediation_router.py)"
        H --> I["POST /v1/remediation/{id}/plan"]
    end
    I --> J[RemediationPlan JSON]
```

Step 1: Generate a remediation plan with balanced priorities

```
curl -s -X POST "http://localhost:8100/v1/remediation/$SCAN_ID/plan" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "objectives": {
      "risk_reduction_weight": 0.5,
      "downtime_weight": 0.3,
      "cost_weight": 0.2
    },
    "constraints": {
      "max_downtime_hours": 4.0,
      "max_cost_units": 50,
      "exclude_action_types": []
    },
    "scope": "all_devices"
  }' | jq .
```

What you should see:

```
{
  "status": "success",
  "data": {
    "plan_id": "plan-a1b2c3d4",
    "created_at": "2026-03-05T16:30:00Z",
    "total_actions": 12,
    "estimated_brs_reduction": 2.8,
    "estimated_downtime_hours": 3.2,
    "estimated_cost_units": 38,
    "pareto_rank": 1,
    "actions": [
      {
        "action_id": "act-001",
        "action_type": "rotate_credentials",
        "target_device": "172.30.0.10",
        "priority": 1,
        "brs_delta": -0.8,
        "downtime_hours": 0.1,
        "cost_units": 1,
        "dependencies": []
      },
      ...
    ]
  }
}
```

Step 2: Save the plan ID for subsequent exercises

```
PLAN_ID=$(curl -s -X POST
        "http://localhost:8100/v1/remediation/$SCAN_ID/plan" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"objectives":
        {"risk_reduction_weight":0.5,"downtime_weight":0.3,"cost_weight":0.2},"constraints":
        {"max_downtime_hours":4.0,"max_cost_units":50},"scope":"all_devices"}' \
  | jq -r '.data.plan_id')

echo "Plan ID: $PLAN_ID"
```

Step 3: Generate a risk-first plan for comparison

```
# Plan that maximizes risk reduction regardless of cost or downtime
curl -s -X POST "http://localhost:8100/v1/remediation/$SCAN_ID/plan" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"objectives":
        {"risk_reduction_weight":0.9,"downtime_weight":0.05,"cost_weight":0.05},"constraints
        {"max_downtime_hours":24.0,"max_cost_units":200},"scope":"all_devices"}' | jq '{
```

```
    plan_id: .data.plan_id,
    brs_reduction: .data.estimated_brs_reduction,
    downtime: .data.estimated_downtime_hours,
    cost: .data.estimated_cost_units,
    actions: .data.total_actions
  }'
```

Step 4: Retrieve and examine the plan structure

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID"
       \
  -H "Authorization: Bearer $TOKEN" | jq '.data.actions |
       group_by(.action_type) | map({type: .[0].action_type, count:
       length, total_brs_delta: (map(.brs_delta) | add)})'
```

Discussion Questions:

1. The Pareto optimizer generates a plan with weights 0.5/0.3/0.2 (risk/downtime/cost). If you change the weights to 0.9/0.05/0.05 (risk-dominated), the plan includes more actions but with higher downtime. Why does prioritizing risk reduction lead to more downtime, not just different action types?

2. The first action in the plan is `rotate_credentials` with `brs_delta: -0.8` and `cost_units: 1`. Why is credential rotation the first action in almost every Pareto-optimal plan, regardless of objective weights?

3. `pareto_rank: 1` indicates this plan is on the Pareto frontier. What would `pareto_rank: 2` mean, and when would a security team choose a rank-2 plan over a rank-1 plan?

4. The plan has `dependencies: []` for the first action. Why do some remediation actions have dependencies, and what happens if a dependent action's prerequisite fails mid-plan?

> **Troubleshooting:** If `total_actions: 0`, no remediable issues were found. Ensure the scan completed with CVE assessment and default credential detection. If plan generation times out (> 30 seconds), reduce scope: `"scope": "critical_devices_only"`.

Exercise 2: Simulate in Digital Twin (Preflight)

(See Slides 026-035 for digital twin simulation, cascade detection, and safety pipeline)

Before executing any action on production devices, Breakwater simulates the plan in the Phase 6 digital twin. The simulation checks for: service disruption cascades (remediating one device unexpectedly disrupts a dependent service), compliance regressions (an action that increases one control but violates another), and health check failures (the device fails its health check after simulated remediation).

Step 1: Run preflight simulation

```
curl -s -X POST
       "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID/simulate"
       \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "simulation_depth": "full",
    "include_cascade_detection": true,
    "health_check_timeout_seconds": 30
  }' | jq '.data.summary'
```

What you should see:

```
{
  "simulation_status": "PASSED_WITH_WARNINGS",
  "actions_simulated": 12,
```

```
  "actions_safe": 10,
  "actions_warning": 2,
  "actions_blocked": 0,
  "cascade_risks_detected": 1,
  "health_check_failures": 0,
  "compliance_regressions": 0,
  "warnings": [
    {
      "action_id": "act-007",
      "warning_type": "service_disruption",
      "description": "Firmware update on 172.30.0.10 will interrupt
        RTSP stream for 45 seconds",
      "cascade_affected": ["172.30.0.100 (NVR)"],
      "mitigation": "Schedule during low-traffic hours; pre-notify NVR
        to handle stream interruption"
    }
  ]
}
```

Step 2: Check cascade risk details

```
curl -s -X POST
        "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID/simulate"
        \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"simulation_depth":"full","include_cascade_detection":true}' |
        jq '.data.cascade_risks'
```

Step 3: Examine blocked actions (if any)

```
curl -s -X POST
        "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID/simulate"
        \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"simulation_depth":"full","include_cascade_detection":true}' |
        jq '.data.blocked_actions'
```

Step 4: Compare simulation result to plan cost estimate

```
# Actual simulated cost vs. plan estimate
curl -s -X POST
        "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID/simulate"
        \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"simulation_depth":"full","include_cascade_detection":true}' |
        jq '{
    estimated_downtime: .data.estimated_downtime_hours,
    simulated_downtime: .data.simulated_downtime_hours,
    delta_percent: (((.data.simulated_downtime_hours -
        .data.estimated_downtime_hours) /
        .data.estimated_downtime_hours) * 100 | round)
  }'
```

Discussion Questions:

1. The simulation detected a cascade risk: firmware update on the camera interrupts the NVR's RTSP stream. This cascade was not predicted by the plan optimizer. Why does cascade detection require a simulation rather than a static analysis of the plan?

2. `actions_blocked: 0` in this simulation. What categories of actions would cause a block (not just a warning) in the safety pipeline?

3. The simulation reveals the plan's downtime estimate was 10% lower than the actual simulated downtime. Why would the optimizer underestimate downtime, and what is the consequence of this systematic underestimation?

4. The simulation runs in a digital twin, not on production devices. What properties of the digital twin must be true for the simulation results to be trustworthy for production deployment decisions?

> **Troubleshooting:** If simulation fails with "digital twin not available", restart the student lab API with `docker compose restart api` from the `student-lab/` directory and wait for `./wait-for-lab.sh` to pass. If cascade detection takes > 60 seconds, use `"simulation_depth": "quick"` (skips multi-hop cascade detection).

Exercise 3: Approve Plan (Manual Gate)

(See Slides 036-042 for human approval workflow, four-eyes principle, and audit trail)

The approval gate is a mandatory checkpoint before any production action. It implements the four-eyes principle: the plan was generated by the system, and a human must explicitly confirm they understand the plan, have reviewed the simulation results, and accept responsibility for execution.

Step 1: Submit plan approval

```
curl -s -X POST
        "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID/approve"
        \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "approved_by": "student@example.com",
    "approval_notes": "Reviewed simulation results. Cascade warning
        acknowledged: scheduling camera firmware update during 2-4 AM
        maintenance window.",
    "accepted_risks": ["act-007-cascade-warning"],
    "override_blocked": false
  }' | jq .
```

What you should see:

```
{
  "status": "success",
  "data": {
    "plan_id": "plan-a1b2c3d4",
    "approval_status": "APPROVED",
    "approved_by": "student@example.com",
    "approved_at": "2026-03-05T16:35:00Z",
    "approval_token": "apprv-9z8y7x6w",
    "execution_authorized_until": "2026-03-06T16:35:00Z",
    "accepted_risks_count": 1,
    "audit_entry_id": "aud-001"
  }
}
```

Step 2: Verify approval is recorded in the audit trail

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/audit" \
  -H "Authorization: Bearer $TOKEN" | jq '.data.entries[] |
        select(.event_type == "plan_approved")'
```

Step 3: Check approval expiry

```
# Approval is valid for 24 hours; execution must start within this
        window
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID"
        \
  -H "Authorization: Bearer $TOKEN" | jq '{approval_status:
        .data.approval_status, valid_until:
        .data.execution_authorized_until}'
```

Step 4: Attempt to modify the plan after approval (should be rejected)

```
# Plans are immutable after approval
curl -s -X PATCH
        "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID"
        \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"actions": []}' | jq '{status: .status, error: .detail}'
```

Discussion Questions:

1. The approval token expires in 24 hours (`execution_authorized_until`). Why is a time-limited approval token necessary? What operational scenario would require a longer validity window, and what security risk does extension introduce?

2. `accepted_risks: ["act-007-cascade-warning"]` -- the approver explicitly accepts the cascade risk for action 7. What is the legal and operational significance of explicitly enumerating accepted risks in the approval record?

3. `override_blocked: false` -- the approver did not override any blocked actions. Under what emergency circumstances would `override_blocked: true` be appropriate, and what additional safeguards should be required for an override?

4. Plans are immutable after approval. Why is this important for the audit trail? What would the consequences be if an approved plan could be silently modified before execution?

> **Troubleshooting:** If approval returns 400 "simulation not completed", run Exercise 2 first (simulation is required before approval). If approval returns 403 "insufficient privileges", the lab user account may need the `security_operator` role -- check with your instructor.

Exercise 4: Execute Credential Rotation

(See Slides 043-050 for credential rotation mechanics, protocol-specific rotation, and rollback)

Step 1: Execute credential rotation for a specific device

```
curl -s -X POST
        "http://localhost:8100/v1/remediation/$SCAN_ID/credentials" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "target_device": "172.30.0.10",
    "protocols": ["http", "rtsp", "onvif"],
    "new_credential_policy": {
      "length": 24,
      "complexity": "high",
      "rotation_interval_days": 90
    },
    "plan_id": "'$PLAN_ID'",
    "dry_run": false
  }' | jq '.data'
```

What you should see:

```
{
  "device_ip": "172.30.0.10",
  "rotation_id": "rot-001",
  "status": "completed",
  "protocols_rotated": ["http", "rtsp", "onvif"],
  "new_credential_hash": "sha256:abc123...",
  "previous_credential_hash": "sha256:000000...",
  "rotation_completed_at": "2026-03-05T16:40:00Z",
```

```
  "health_check_passed": true,
  "brs_delta_applied": -0.8,
  "rollback_available": true,
  "rollback_expires_at": "2026-03-05T17:10:00Z"
}
```

Step 2: Verify the device remains accessible after rotation

```
# Health check confirms device is still responsive after credential
        rotation
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/credentials" \
  -H "Authorization: Bearer $TOKEN" | jq '.data.rotations[] |
        select(.device_ip == "172.30.0.10") | {status: .status,
        health: .health_check_passed}'
```

Step 3: Check rollback availability

```
# Rollback is available for 30 minutes after rotation
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/credentials" \
  -H "Authorization: Bearer $TOKEN" | jq '.data.rotations[0] |
        {rollback_available: .rollback_available, expires:
        .rollback_expires_at}'
```

Step 4: Attempt credential rotation in dry-run mode for another device

```
# Dry run shows what would happen without actually rotating
curl -s -X POST
        "http://localhost:8100/v1/remediation/$SCAN_ID/credentials" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "target_device": "172.30.0.11",
    "protocols": ["http"],
    "new_credential_policy": {"length": 20, "complexity": "high"},
    "plan_id": "'$PLAN_ID'",
    "dry_run": true
  }' | jq '{dry_run: .data.dry_run, would_rotate_protocols:
        .data.protocols_to_rotate, estimated_impact:
        .data.estimated_impact}'
```

Discussion Questions:

1. The rotation records the previous credential as a hash (`previous_credential_hash: sha256:000000...`). Why do all-zeros suggest the previous credential was the default factory credential, and what does this confirm about the device's prior security state?

2. `rollback_available: true` for 30 minutes. If a credential rotation breaks a dependent system (e.g., the NVR loses connection to the camera because it stored the old credential), what is the rollback procedure and what state will each system be in after rollback?

3. The system rotates credentials for three protocols simultaneously (HTTP, RTSP, ONVIF). Why is simultaneous rotation preferable to sequential rotation? What race condition could occur with sequential rotation?

4. After credential rotation, the `brs_delta_applied: -0.8` is recorded. How does this value get incorporated into the next scan's BRS score, and what would happen to the BRS if the rotation were rolled back?

> **Troubleshooting:** If rotation returns `health_check_passed: false`, the device may not have accepted the new credential. This is expected for some simulated devices that reject credential changes. The rotation is automatically rolled back. Use `dry_run: true` to test without affecting device state.

Exercise 5: Stage Firmware Update

(See Slides 051-058 for firmware update staging, canary deployment, rollback, and integrity verification)

Step 1: Initiate a staged firmware update

```
curl -s -X POST
        "http://localhost:8100/v1/remediation/$SCAN_ID/firmware" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "target_device": "172.30.0.10",
    "firmware_version": "latest_stable",
    "staging_strategy": "canary",
    "canary_percent": 10,
    "rollback_on_failure": true,
    "pre_update_snapshot": true,
    "plan_id": "'$PLAN_ID'"
  }' | jq '.data'
```

What you should see:

```
{
  "update_id": "fw-001",
  "device_ip": "172.30.0.10",
  "current_version": "5.2.0.build220922",
  "target_version": "5.7.1.build260105",
  "staging_strategy": "canary",
  "canary_percent": 10,
  "status": "staging",
  "stages": [
    {"stage": "pre_snapshot", "status": "completed", "timestamp":
        "..."},
    {"stage": "canary_deploy", "status": "in_progress", "timestamp":
        "..."},
    {"stage": "canary_health_check", "status": "pending"},
    {"stage": "full_deploy", "status": "pending"},
    {"stage": "post_verification", "status": "pending"}
  ],
  "rollback_snapshot_id": "snap-001",
  "estimated_completion_minutes": 8
}
```

Step 2: Monitor the update progress

```
UPDATE_ID="fw-001"  # Use the actual update_id from Step 1
sleep 5  # Wait for staging to progress

curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/firmware" \
  -H "Authorization: Bearer $TOKEN" | jq '.data.updates[] | {id:
        .update_id, device: .device_ip, status: .status,
        current_stage: (.stages[] | select(.status == "in_progress") |
        .stage)}'
```

Step 3: Verify firmware integrity post-update

```
# Check that the installed firmware hash matches the vendor's expected
        hash
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/firmware" \
  -H "Authorization: Bearer $TOKEN" | jq '.data.updates[0] | {
    installed_hash: .installed_firmware_hash,
    expected_hash: .vendor_expected_hash,
    hash_verified: .integrity_verified
  }'
```

Step 4: Check rollback snapshot is available

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/firmware" \
  -H "Authorization: Bearer $TOKEN" | jq '.data.updates[0] | {
    rollback_available: .rollback_snapshot_id,
    rollback_expires: .rollback_snapshot_expires_at
  }'
```

Discussion Questions:

1. The staging strategy is "canary" with `canary_percent: 10`. With only 1 camera in the lab, 10% would mean 0.1 cameras -- which rounds to 0. How does the canary strategy adapt for small fleets, and at what fleet size does canary deployment become operationally meaningful?

2. The firmware update jumps from version 5.2.0 to 5.7.1 -- skipping 4 minor versions. What risk does "version skipping" introduce, and how should the update staging strategy handle a detected compatibility gap?

3. `pre_update_snapshot: true` creates a device state snapshot before the update. What does a firmware snapshot include for an embedded device, and what are the limitations of restoring from a firmware snapshot vs. a full factory reset?

4. `integrity_verified: true` confirms the installed firmware hash matches the vendor's expected hash. But the "expected hash" was retrieved from the vendor's server. What trust assumptions does this make, and how could the supply chain be compromised even with hash verification?

> **Troubleshooting:** If firmware update returns "vendor firmware not available", the IoT sim devices use synthetic firmware version numbers that may not have a real firmware binary. The update will simulate the stage progression but skip the actual binary transfer. This is expected lab behavior -- the stages and health checks still demonstrate the workflow.

Exercise 6: Generate Segmentation Policy

(See Slides 059-066 for network segmentation, iptables/nftables format, VLAN policy, and zone isolation)

Step 1: Generate the network segmentation policy

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/segmentation" \
  -H "Authorization: Bearer $TOKEN" | jq '.data.summary'
```

What you should see:

```
{
  "policy_id": "seg-001",
  "zones_defined": 4,
  "rules_generated": 47,
  "policy_format": "iptables",
  "estimated_risk_reduction_percent": 65,
  "zones": [
    {"zone": "camera", "devices": 8, "allowed_destinations": ["nvr-
        zone", "management"]},
    {"zone": "storage", "devices": 3, "allowed_destinations":
        ["management"]},
    {"zone": "network-infrastructure", "devices": 2,
        "allowed_destinations": ["all-zones"]},
    {"zone": "iot-sensors", "devices": 7, "allowed_destinations":
        ["cloud-endpoint"]}
  ]
}
```

Step 2: View the generated iptables rules

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/segmentation" \
  -H "Authorization: Bearer $TOKEN" | jq -r '.data.iptables_rules | .
        [:10] | .[]'
```

What you should see (first 10 rules):

```
# Breakwater Phase 12 Segmentation Policy
# Generated: 2026-03-05T16:45:00Z
# Zone: camera -> nvr-zone (RTSP, HTTP only)
-A FORWARD -s 172.30.0.10/32 -d 172.30.0.100/32 -p tcp --dport 554 -j
ACCEPT
-A FORWARD -s 172.30.0.10/32 -d 172.30.0.100/32 -p tcp --dport 80 -j
ACCEPT
-A FORWARD -s 172.30.0.10/32 -d 172.30.0.100/32 -p tcp --dport 443 -j
ACCEPT
# Camera-to-camera lateral movement blocked (default deny)
-A FORWARD -s 172.30.0.10/32 -d 172.30.0.11/32 -j DROP
-A FORWARD -s 172.30.0.10/32 -d 172.30.0.11/32 -j LOG --log-prefix
"BW-CAM-LATERAL: "
...
```

Step 3: Get nftables format (modern Linux firewall)

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/segmentation?
        format=nftables" \
  -H "Authorization: Bearer $TOKEN" | jq -r '.data.nftables_rules | .
        [:5] | .[]'
```

Step 4: Estimate segmentation impact on BRS scores

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/segmentation" \
  -H "Authorization: Bearer $TOKEN" | jq '.data.brs_impact_by_device |
        to_entries | sort_by(-.value.new_brs) | .[:5] | map({ip: .key,
        old_brs: .value.old_brs, new_brs: .value.new_brs, improvement:
        (.value.old_brs - .value.new_brs | . * 100 | round / 100)})'
```

Discussion Questions:

1. The policy blocks camera-to-camera lateral movement (`172.30.0.10 -> 172.30.0.11 DROP`). All cameras are on the same /24 subnet. What infrastructure change is required to actually enforce this rule -- is iptables on the scanner sufficient, or does the network hardware need configuration?

2. The policy allows cameras to reach only the NVR-zone on specific ports (554, 80, 443). What legitimate camera function might be broken by this policy, and how would you detect this break during the simulation phase?

3. `estimated_risk_reduction_percent: 65` -- segmentation reduces risk by 65%. But the remaining 35% of risk persists because segmentation doesn't fix vulnerabilities, only limits lateral movement. Identify the top 3 risk categories that segmentation cannot address.

4. The policy is in iptables format. In a real enterprise environment, you would not implement iptables rules on every device -- you would push policy to network equipment (switches, routers, SDN controller). How would you adapt the generated iptables rules for a Cisco VLAN ACL, an AWS Security Group, or an SDN controller API?

> **Troubleshooting:** If `zones_defined: 0`, segmentation requires device type information from Phase 1 identification. If device types are "unknown", segmentation cannot create zones. Re-run the scan and ensure fingerprinting completed.

Exercise 7: Check Compliance Before (IEC 62443 Score)

(See Slides 067-074 for IEC 62443 control families, Security Level assessment, and maturity scoring)

Step 1: Get the pre-remediation IEC 62443 compliance score

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/compliance/before"
        \
  -H "Authorization: Bearer $TOKEN" | jq '.data'
```

What you should see:

```
{
  "standard": "IEC 62443-3-3",
  "assessment_timestamp": "2026-03-05T16:00:00Z",
  "overall_security_level": "SL-1",
  "target_security_level": "SL-2",
  "overall_score": 38.5,
  "control_families": {
    "FR1_identification_authentication": {"score": 25.0, "max": 100,
        "gaps": 3},
    "FR2_use_control": {"score": 45.0, "max": 100, "gaps": 2},
    "FR3_system_integrity": {"score": 40.0, "max": 100, "gaps": 4},
    "FR4_data_confidentiality": {"score": 30.0, "max": 100, "gaps":
        3},
    "FR5_restricted_data_flow": {"score": 20.0, "max": 100, "gaps":
        5},
    "FR6_timely_response": {"score": 55.0, "max": 100, "gaps": 2},
    "FR7_resource_availability": {"score": 60.0, "max": 100, "gaps":
        1}
  },
  "critical_gaps": [
    {"control": "SR-1.1", "description": "Human user identification
        and authentication", "gap": "Default credentials on 8
        devices"},
    ...
  ]
}
```

Step 2: Identify the lowest-scoring control family

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/compliance/before"
        \
  -H "Authorization: Bearer $TOKEN" | jq '.data.control_families |
        to_entries | sort_by(.value.score) | .[0]'
```

Step 3: List all critical compliance gaps

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/compliance/before"
        \
  -H "Authorization: Bearer $TOKEN" | jq '.data.critical_gaps[] |
        {control: .control, description: .description}'
```

Step 4: Map compliance gaps to remediation actions in the plan

```
# See which plan actions address which compliance gaps
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID"
        \
  -H "Authorization: Bearer $TOKEN" | jq '.data.actions[] |
        {action_type: .action_type, compliance_controls_addressed:
        .compliance_controls_addressed}'
```

Discussion Questions:

1. The overall security level is SL-1 with a target of SL-2. What is the difference between Security Level 1 and Security Level 2 in IEC 62443 terms, and which IEC 62443-3-3 requirements must be met to advance from SL-1 to SL-2?

2. FR5 (Restricted Data Flow) has the lowest score (20/100). This control family maps directly to network segmentation. How does the segmentation policy from Exercise 6 specifically address FR5, and which FR5 controls will still require manual implementation after the policy is applied?

3. The compliance assessment identifies SR-1.1 (Human User Identification and Authentication) as a critical gap because "Default credentials on 8 devices." After credential rotation (Exercise 4), which SR-1.1 sub-controls will be satisfied, and which will remain gaps?

4. IEC 62443 is an industrial cybersecurity standard primarily designed for OT/SCADA environments. The lab environment is IoT (cameras, NAS, sensors). How well does IEC 62443 translate to IoT security assessment, and where do you see gaps in coverage for IoT-specific threats?

> **Troubleshooting:** If compliance shows all N/A, the IEC 62443 assessment requires device classification (OT/IT/IoT) from the scan. If devices are unclassified, the assessment cannot map them to IEC 62443 control families. Set `BREAKWATER_IEC62443_MODE=iot` to use the IoT profile.

Exercise 8: Execute Full Plan and Verify Health

(See Slides 075-080 for plan execution, step-by-step progress, health verification, and abort conditions)

Step 1: Execute the approved remediation plan

```
curl -s -X POST
        "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID/execute"
        \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "execution_mode": "sequential",
    "abort_on_health_failure": true,
    "pause_between_actions_seconds": 5
  }' | jq '.data.execution_id'
```

Step 2: Monitor execution progress

```
EXEC_ID=$(curl -s -X POST
        "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID/execute"
        \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"execution_mode":"sequential","abort_on_health_failure":true,"pause_between_actions_s
  | jq -r '.data.execution_id')

# Poll for progress
sleep 10
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID"
        \
  -H "Authorization: Bearer $TOKEN" | jq '{
    execution_status: .data.execution_status,
    actions_completed: .data.actions_completed,
    actions_remaining: .data.actions_remaining,
    health_failures: .data.health_failures
  }'
```

Step 3: Check health of each device post-remediation

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID"
        \
  -H "Authorization: Bearer $TOKEN" | jq
        '.data.device_health_post_execution | to_entries | map({ip:
        .key, status: .value.status, services_up:
        .value.services_responding})'
```

Step 4: Verify BRS improvement after execution

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/plan/$PLAN_ID"
        \
  -H "Authorization: Bearer $TOKEN" | jq '{
    planned_brs_reduction: .data.estimated_brs_reduction,
    actual_brs_reduction: .data.actual_brs_reduction,
```

```
    deviation_percent: (((.data.actual_brs_reduction -
        .data.estimated_brs_reduction) / .data.estimated_brs_reduction
        * 100) | round)
  }'
```

Discussion Questions:

1. `execution_mode: "sequential"` executes one action at a time. An alternative is `"parallel"` (execute all actions simultaneously). When would you prefer parallel execution, and what risk does parallel execution introduce that sequential execution avoids?

2. `abort_on_health_failure: true` -- if any device fails its post-action health check, the entire plan stops. What is the tradeoff between aborting immediately (safer, more conservative) and continuing past failures (faster, but risks compounding failures)?

3. The actual BRS reduction differs from the planned estimate. What factors could cause the actual reduction to be higher than estimated? What factors could cause it to be lower?

4. After the plan completes, some actions may have succeeded while others failed. How does the system maintain a consistent security state when a partially executed plan leaves some devices in the new state and some in the old state?

> **Troubleshooting:** If execution returns 403 "plan not approved", complete Exercise 3 first. If execution aborts with "health check timeout", increase the health check timeout: `"health_check_timeout_seconds": 60`. If the plan stalls mid-execution, check API logs with `docker compose logs api` from the `student-lab/` directory.

Exercise 9: Check Compliance After (Improvement)

(See Slides 081-086 for compliance improvement measurement, before-after comparison, and certification readiness)

Step 1: Get the post-remediation IEC 62443 score

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/compliance/after/$PLAN_ID"
       \
  -H "Authorization: Bearer $TOKEN" | jq '.data'
```

Step 2: Compare before and after scores

```
BEFORE=$(curl -s
       "http://localhost:8100/v1/remediation/$SCAN_ID/compliance/before"
       \
  -H "Authorization: Bearer $TOKEN" | jq '.data.overall_score')

AFTER=$(curl -s
       "http://localhost:8100/v1/remediation/$SCAN_ID/compliance/after/$PLAN_ID"
       \
  -H "Authorization: Bearer $TOKEN" | jq '.data.overall_score')

echo "Before: $BEFORE"
echo "After: $AFTER"
echo "Improvement: $(echo "$AFTER - $BEFORE" | bc)"
```

Step 3: Check per-control-family improvement

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/compliance/after/$PLAN_ID"
       \
  -H "Authorization: Bearer $TOKEN" | jq '.data.control_families |
       to_entries | map({family: .key, score: .value.score,
       improvement: .value.improvement_from_baseline})'
```

Step 4: Determine if SL-2 target is achieved

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/compliance/after/$PLAN_ID"
      \
  -H "Authorization: Bearer $TOKEN" | jq '{
    security_level: .data.overall_security_level,
    target_met: .data.target_security_level_achieved,
    remaining_gaps: (.data.remaining_critical_gaps | length)
  }'
```

Discussion Questions:

1. The compliance score improved from 38.5 to 52.0 (a 13.5-point improvement). The target SL-2 requires a score of 70.0. What category of remediation actions would be needed to close the remaining 18-point gap, and why did the current plan not address these?

2. FR1 (Identification and Authentication) improved the most after credential rotation and MFA enablement. FR5 (Restricted Data Flow) improved after segmentation. Which control family likely did NOT improve despite the remediation plan, and why?

3. "Certification readiness" for IEC 62443 requires a third-party assessment, not just a self-assessment tool. What is the role of Breakwater's compliance score in a formal IEC 62443 certification process?

4. A competitor tool claims their automated remediation achieves "IEC 62443 SL-3 compliance in one click." What questions would you ask to evaluate this claim, and why is SL-3 through automated tools alone likely misleading?

> **Troubleshooting:** If `after` score equals `before` score, the plan execution may not have completed. Run Exercise 8 first and confirm `execution_status: "completed"`. The after-compliance endpoint reads from the executed plan's results.

Exercise 10: Export Audit Evidence Package

(See Slides 087-098 for audit evidence, do-calculus counterfactuals, regulatory traceability, and long-term archival)

The audit evidence package provides a cryptographically signed record of everything that happened during the remediation cycle: the risk assessment that triggered the plan, the plan itself, simulation results, approval record, execution log, and compliance measurements. This package satisfies regulatory requirements for change management evidence (NERC CIP, NIS2, HIPAA).

Step 1: Export the complete audit package

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/audit" \
  -H "Authorization: Bearer $TOKEN" | jq '.data.summary'
```

What you should see:

```
{
  "audit_package_id": "aud-pkg-001",
  "generated_at": "2026-03-05T17:00:00Z",
  "scan_id": "a3f1c2d4-...",
  "plan_id": "plan-a1b2c3d4",
  "sections_included": [
    "risk_assessment", "remediation_plan", "simulation_results",
    "approval_record", "execution_log", "compliance_before",
        "compliance_after",
    "counterfactual_analysis"
  ],
  "total_events": 87,
  "package_hash": "sha256:9f8e7d6c...",
  "chain_valid": true,
  "signing_key_id": "bw-audit-key-2026-q1",
```

```
  "regulatory_mappings": ["NERC CIP-007", "NIS2 Art.21", "IEC 62443"]
}
```

Step 2: Examine the counterfactual analysis section

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/audit" \
  -H "Authorization: Bearer $TOKEN" | jq
        '.data.counterfactual_analysis'
```

What you should see:

```
{
  "analysis_method": "do-calculus Pearl causal model",
  "question": "What would have happened had we NOT performed
        remediation?",
  "counterfactual_breach_probability_no_remediation": 0.34,
  "counterfactual_breach_probability_with_remediation": 0.09,
  "expected_loss_avoidance_usd": 142000,
  "confidence_interval": "90%: [$80,000 - $220,000]"
}
```

Step 3: Get regulatory traceability for each action

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/audit" \
  -H "Authorization: Bearer $TOKEN" | jq
        '.data.regulatory_traceability[] | {control:
        .regulatory_control, action: .remediation_action, evidence:
        .evidence_reference}'
```

Step 4: Export in OSCAL (Open Security Controls Assessment Language) format

```
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/audit?
        format=oscal" \
  -H "Authorization: Bearer $TOKEN" | jq '{
    oscal_version: .data.oscal_version,
    component_count: (.data.components | length),
    control_implementations: (.data.control_implementations | length)
  }'
```

Discussion Questions:

1. The counterfactual analysis uses Pearl's do-calculus to answer: "What would have happened without remediation?" How does causal inference (do-calculus) differ from standard statistical correlation, and why is the distinction important for justifying remediation investments to a CFO?

2. `expected_loss_avoidance_usd: 142,000` with 90% confidence interval `[$80,000 - $220,000]`. The remediation plan cost was $38,000. How would you present this to a budget committee to justify the $38,000 investment?

3. OSCAL (Open Security Controls Assessment Language) is a NIST standard for machine-readable security assessment. What advantage does OSCAL format provide over PDF audit reports for regulatory submission?

4. The audit package has `chain_valid: true` and is signed by `bw-audit-key-2026-q1`. For long-term archival (10+ years), what cryptographic considerations must be addressed for the signature to remain verifiable, and what does this require from the audit infrastructure?

> **Troubleshooting:** If counterfactual analysis shows null, the do-calculus model requires breach probability estimates from Phase 4 attack paths. Ensure Phase 4 was computed. If OSCAL format returns 404, OSCAL export requires `OSCAL_ENABLED=true` environment variable.

Lab Wrap-Up

You have completed the full Breakwater remediation lifecycle across 10 exercises:

1. **Generated** a Pareto-optimized remediation plan balancing risk, downtime, and cost
2. **Simulated** the plan in the digital twin to detect cascade risks before production impact
3. **Approved** the plan through a mandatory human gate with risk acknowledgment
4. **Rotated** credentials autonomously with rollback capability and BRS delta tracking
5. **Staged** a firmware update with canary strategy, integrity verification, and health checks
6. **Generated** an iptables/nftables segmentation policy reducing lateral movement by 65%
7. **Measured** IEC 62443 compliance baseline (SL-1, score 38.5) before remediation
8. **Executed** the full plan with health monitoring, abort conditions, and BRS verification
9. **Confirmed** compliance improvement (to 52.0) and remaining gaps to SL-2 target
10. **Exported** a cryptographically signed audit package with do-calculus counterfactuals

The Breakwater platform's final promise is this: every security action is preceded by a proof (simulation), authorized by a human (approval gate), executed safely (health checks, rollback), and measured for impact (BRS and compliance delta). Every action is recorded in a tamper-evident, cryptographically signed audit trail that satisfies regulatory requirements. Security operations move from reactive incident response to autonomous, continuous, evidence-driven risk reduction.

Final status check:

```
# Get the final state of the entire remediation lifecycle
curl -s "http://localhost:8100/v1/remediation/$SCAN_ID/audit" \
  -H "Authorization: Bearer $TOKEN" | jq '{
    plan_id: .data.plan_id,
    approval_status: .data.approval_status,
    execution_complete: .data.execution_completed,
    compliance_before: .data.compliance_before_score,
    compliance_after: .data.compliance_after_score,
    audit_package_id: .data.audit_package_id,
    chain_valid: .data.chain_valid
  }'
```