SEAS-8414 Week 08 Student Lab Guide

SEAS-8414 Week 08 Student Lab Guide: Federated Threat Intelligence Network

Start Here: Download the Student ZIP

Before running any lab commands, download the Week 08 student package from the course site:

https://8414.bwater.io/downloads/labs/packages/seas8414-blackboard-week-08-2026.05.0_4e76a52f_aws8414.zip

The recommended path is the package Makefile:

```
unzip seas8414-blackboard-week-08-2026.05.0_4e76a52f_aws8414.zip
cd seas8414-blackboard-week-08-2026.05.0_4e76a52f_aws8414
make week08
```

The Makefile calls `run-week08-lab.sh`, extracts the nested runtime ZIP, starts the lab, runs the Week 08 API workflow, saves evidence under `lab-results/week-08/evidence/`, generates `lab-results/week-08/index.html`, and cleans up containers at exit.

You can also download the runner directly from https://8414.bwater.io/downloads/labs/scripts/run-week08-lab.sh.

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

```
unzip seas8414-blackboard-week-08-2026.05.0_4e76a52f_aws8414.zip
cd seas8414-blackboard-week-08-2026.05.0_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/phase08-lab-screencast.mp4
- LLM Prompt: https://8414.bwater.io/downloads/labs/prompts/phase08-lab-llm-prompt.md
- Run Script: https://8414.bwater.io/downloads/labs/scripts/run-week08-lab.sh

---

Breakwater Phase 8: Federated Threat Intelligence Network Lab

Phase 8 introduces a federated learning system for collaborative threat intelligence sharing: local anomaly model training with differential privacy guarantees, gradient privatisation and secure aggregation (FedAvg), adversarial robustness via randomised smoothing, Byzantine-fault-tolerant aggregation (Krum selection), causal attribution analysis, and a reinforcement learning threat hunting agent. These capabilities enable organisations to share threat intelligence without exposing raw scan data to a central authority.

(See Slides 001-005 for Phase 8 overview, federated learning architecture, and privacy threat model)

Prerequisites

- Completed Phase 1 lab (scan data available in the lab environment)
- Completed Phase 4 lab (attack graph and BRS scores computed)
- A completed scan with `$SCAN_ID` and `$TOKEN` variables set (see Phase 1 Exercise 6)
- Basic understanding of machine learning concepts (training, loss, accuracy)
- Familiarity with privacy concepts (differential privacy, epsilon budget)

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`, no completed scan exists -- run `python scan_report.py 172.30.0.0/24` to create one.

Phase 8 API Cheatsheet

Endpoint	Method	Description
`/v1/federated/{scan_id}/train`	GET	Local federated training rounds and metrics
`/v1/federated/{scan_id}/signals`	GET	Anonymised threat signals from local scan
`/v1/federated/{scan_id}/model`	GET	Latest federated model status and accuracy
`/v1/federated/aggregate`	POST	Trigger FedAvg aggregation across sites
`/v1/federated/{scan_id}/threats`	GET	Federated threat signal matches and analysis
`/v1/federated/{scan_id}/privacy`	GET	Differential privacy budget accounting

All Phase 8 endpoints require Bearer token authentication.

(See Slides 081-090 for API design, endpoint reference, and database models)

Exercise 1: Train the Local Anomaly Model

(See Slides 010-022 for federated learning problem statement, model architecture, and training pipeline)

In federated learning, each participating site trains a local model on its own data and then shares only the model gradients (not the raw data) with a central aggregator. Breakwater's local model is an Isolation Forest or Transformer-based anomaly detector that learns the baseline behaviour of the local network's devices.

Architecture: What This Exercise Tests

```
graph LR
    subgraph "Local Training (fl_trainer.py)"
```

```
        A[Scan Host Data] --> B[FeatureExtractor]
        B --> C[Feature Matrix<br/>N hosts x K features]
        C --> D[IsolationForest /<br/>TransformerIDS]
        D --> E[LocalModelUpdate]
    end
    subgraph "Differential Privacy (dp_engine.py)"
        E --> F[Gradient Clipping<br/>clip_norm=1.0]
        F --> G[Gaussian Noise<br/>calibrated to epsilon]
        G --> H[GradientPayload]
    end
    subgraph "API"
        H --> I["GET /v1/federated/{scan_id}/train"]
    end
```

Step 1: Retrieve local training rounds

```bash
# Get all training rounds for this scan
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" | jq .

# Summary: number of rounds and latest metrics
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{total_rounds: .data.total_rounds,
         latest_round: .data.rounds[-1]}'
```

What you should see:

```
{
  "total_rounds": 3,
  "latest_round": {
    "round_number": 3,
    "site_id": "site-local",
    "loss": 0.142,
    "accuracy": 0.923,
    "sample_count": 20,
    "epsilon_spent": 0.5,
    "gradient_norm": 0.87,
    "clipped": false
  }
}
```

Step 2: Examine training progression across rounds

```bash
# Loss and accuracy across all rounds
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.rounds[] | {round: .round_number, loss, accuracy,
         epsilon_spent}]'
```

```
# Total epsilon spent across all rounds
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" \
  | jq '(.data.rounds | map(.epsilon_spent) | add) as $total |
        {total_epsilon_spent: $total, rounds: .data.total_rounds,
         avg_epsilon_per_round: ($total / .data.total_rounds)}'
```

What you should see for training progression:

```
[
  { "round": 1, "loss": 0.312, "accuracy": 0.781, "epsilon_spent": 0.5
        },
  { "round": 2, "loss": 0.218, "accuracy": 0.872, "epsilon_spent": 0.5
        },
  { "round": 3, "loss": 0.142, "accuracy": 0.923, "epsilon_spent": 0.5
        }
]
```

Step 3: Inspect gradient privacy properties

```
# Check if any gradients were clipped (privacy protection active)
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.rounds[] | {round: .round_number, gradient_norm,
         clipped: (.gradient_norm > 1.0)}]'
```

What you should see:

```
[
  { "round": 1, "gradient_norm": 1.24, "clipped": true  },
  { "round": 2, "gradient_norm": 0.98, "clipped": false },
  { "round": 3, "gradient_norm": 0.87, "clipped": false }
]
```

Round 1's gradient was clipped because its L2 norm (1.24) exceeded the clip bound (1.0). Clipping is the first step of differential privacy: it bounds how much any single training sample can influence the gradient.

> **Troubleshooting:** If `total_rounds: 0` or the endpoint returns 404, the federated learning phase did not run for this scan. Phase 8 requires at least one scan to have been stored in the federated DB. Verify: `curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals -H "Authorization: Bearer $TOKEN" | jq '.data.total_signals'` -- if zero, the feature extraction step produced no signals.

Questions:

1. Training loss decreased from 0.312 to 0.142 across 3 rounds while accuracy improved from 78.1% to 92.3%. Is this improvement rate typical for anomaly detection models? How many training samples (network hosts) were used? Would you expect better accuracy with more hosts? *(See Slides 013-016 for the training pipeline and convergence behaviour)*

2. The gradient clip norm is 1.0. Round 1 had `gradient_norm: 1.24`, which was clipped. What specifically is being clipped -- the entire gradient vector, or individual components? After clipping, what is the gradient norm? *(See Slide 019 for gradient clipping mechanics)*
3. Each training round consumes 0.5 epsilon of privacy budget. With a total budget of 10.0 epsilon, how many training rounds can be run before the budget is exhausted? What happens after exhaustion -- can the model still be used, or must training stop? *(See Slide 020 for privacy budget management)*
4. The local model uses an Isolation Forest architecture. Explain conceptually how Isolation Forest detects anomalies: what does it mean for a data point to be "isolated" quickly? Why is this architecture suitable for network anomaly detection where normal behaviour is the majority? *(See Slides 014-015 for Isolation Forest architecture and IoT anomaly detection)*

Exercise 2: Extract Threat Signals

(See Slides 023-033 for threat signal taxonomy, feature extraction, and anonymisation pipeline)

Threat signals are observations extracted from a scan that indicate potential adversarial behaviour. Before sharing with the federated network, signals are anonymised: IP addresses are replaced with hashed site-specific pseudonyms, and raw port numbers are replaced with service category labels. The five signal types are: `new_port`, `service_change`, `timing_anomaly`, `credential_spray`, and `lateral_movement`.

Architecture: What This Exercise Tests

```
graph LR
    subgraph "Feature Extractor (feature_extractor.py)"
        A[Scan Host Data] --> B[BehavioralFeature]
        B --> C1[port_vector<br/>binary 65K bits]
        B --> C2[service_entropy<br/>Shannon entropy]
        B --> C3[timing_features<br/>response latency]
        B --> C4[cred_attempt_count]
    end
    subgraph "Signal Extractor (threat_signals.py)"
        C1 & C2 & C3 & C4 --> D{Anomaly threshold}
        D -->|port not in baseline| E["new_port signal"]
        D -->|service changed| F["service_change signal"]
        D -->|timing deviation| G["timing_anomaly signal"]
        D -->|spray pattern| H["credential_spray signal"]
        D -->|lateral hop| I["lateral_movement signal"]
    end
    subgraph "Anonymisation"
        E & F & G & H & I --> J[hash(ip + site_secret)]
        J --> K["GET /v1/federated/{scan_id}/signals"]
    end
```

Step 1: Retrieve anonymised threat signals

```
# All threat signals from this scan
curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals \
  -H "Authorization: Bearer $TOKEN" | jq .

# Summary
curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{total_signals: .data.total_signals,
         signal_types: [.data.signals[].signal_type] | group_by(.) |
        map({type: .[0], count: length})}'
```

What you should see:

```
{
  "total_signals": 12,
  "signal_types": [
    { "type": "credential_spray",  "count": 4 },
    { "type": "lateral_movement",  "count": 3 },
    { "type": "new_port",          "count": 3 },
    { "type": "service_change",    "count": 1 },
    { "type": "timing_anomaly",    "count": 1 }
  ]
}
```

Step 2: Examine individual signals

```
# List all signals with details
curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.signals[] | {signal_id, signal_type, severity,
        confidence,
         source_site, anonymized_features}]'

# Filter for high/critical severity signals
curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.signals[] | select(.severity == "high" or .severity ==
        "critical")
        | {signal_type, severity, confidence, anonymized_features}]'
```

What you should see:

```
[
  {
    "signal_id": "sig-a3f1c2d4",
    "signal_type": "credential_spray",
    "severity": "high",
    "confidence": 0.92,
    "source_site": "site-7f3a",
```

```
    "anonymized_features": {
      "affected_device_hash": "3d4e5f6a7b8c",
      "protocol": "http",
      "attempt_count": 20,
      "unique_credential_count": 8
    }
  },
  {
    "signal_id": "sig-b5e6f789",
    "signal_type": "lateral_movement",
    "severity": "critical",
    "confidence": 0.87,
    "source_site": "site-7f3a",
    "anonymized_features": {
      "hop_count": 3,
      "shared_credential": true,
      "device_type_sequence": ["camera", "camera", "nvr"]
    }
  }
]
```

Note that `source_site` is a hashed pseudonym (`site-7f3a`) rather than the real site identifier, and `affected_device_hash` replaces the actual IP address.

Step 3: Analyse signal confidence distribution

```
# Confidence histogram
curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.signals[] | .confidence] | sort |
        { min: .[0], max: .[-1], avg: (add / length) }'

# Signals with confidence > 0.8 (high-confidence alerts)
curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.signals[] | select(.confidence > 0.8)
        | {signal_type, severity, confidence}] |
        sort_by(-.confidence)'
```

What you should see:

```
{ "min": 0.61, "max": 0.95, "avg": 0.79 }

[
  { "signal_type": "credential_spray",  "severity": "high",
        "confidence": 0.95 },
  { "signal_type": "credential_spray",  "severity": "high",
        "confidence": 0.92 },
  { "signal_type": "lateral_movement",  "severity": "critical",
        "confidence": 0.87 },
```

```
  { "signal_type": "new_port",          "severity": "medium",
      "confidence": 0.83 }
]
```

> **Troubleshooting:** If `total_signals: 0`, the feature extractor did not find any anomalous patterns. This is expected if the scan has no credential-spray activity or no baseline to compare against (first scan). Run a second scan after the IoT simulator's vulnerable containers have been active for a period to generate timing anomaly signals.

Questions:

1. Which signal type appears most frequently in your scan? What network behaviour produces a `credential_spray` signal, and how does the feature extractor detect it from scan data rather than live packet capture? *(See Slides 026-028 for signal extraction algorithms)*
2. The `anonymized_features` field replaces IP addresses with hashed pseudonyms. Examine the `affected_device_hash` field -- it is a SHA-256 hash of the device IP plus a site-specific secret. What privacy property does this provide? Can the central aggregator correlate signals from two different sites if they affect the same IP range? *(See Slide 031 for anonymisation guarantees and their limitations)*
3. A `lateral_movement` signal has `device_type_sequence: ["camera", "camera", "nvr"]`. This sequence was extracted from Phase 4 attack paths. How does including device type sequences rather than IP sequences improve the utility of the signal for cross-site correlation? What privacy risk does device type inference introduce? *(See Slide 029 for feature selection for privacy-preserving signals)*
4. The confidence field ranges from 0.61 to 0.95. Design a threshold policy for when a signal should be automatically escalated vs. queued for human review. Consider: (a) false positive cost, (b) false negative cost, (c) analyst workload. *(See Slide 033 for signal confidence calibration)*

Exercise 3: Check the Privacy Budget

(See Slides 034-044 for differential privacy theory, Gaussian mechanism, and budget accounting)

Differential privacy (DP) provides a mathematical guarantee: adding or removing any single training sample from the dataset changes the model's output by at most epsilon (with probability 1-delta). The epsilon budget tracks cumulative privacy expenditure across training rounds. When the budget is exhausted, further training would compromise the privacy guarantee.

Architecture: What This Exercise Tests

```
graph LR
    subgraph "DP Engine (dp_engine.py)"
        A[Gradient Vector] --> B[L2 Clip<br/>clip_norm=1.0]
        B --> C[Gaussian Noise<br/>sigma = clip_norm * sqrt(2 *
```

```
ln(1.25/delta)) / epsilon]
        C --> D[Privatised Gradient]
    end
    subgraph "Budget Accounting"
        D --> E["epsilon_spent += dp_epsilon"]
        E --> F[PrivacyBudget]
        F --> G["remaining = total - spent"]
        G --> H["rounds_remaining = remaining / dp_epsilon"]
    end
    subgraph "API"
        F --> I["GET /v1/federated/{scan_id}/privacy"]
    end
```

Step 1: View the current privacy budget

```
# Full privacy budget status
curl -s http://localhost:8100/v1/federated/$SCAN_ID/privacy \
  -H "Authorization: Bearer $TOKEN" | jq .

# Key budget metrics
curl -s http://localhost:8100/v1/federated/$SCAN_ID/privacy \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{total_epsilon: .data.total_epsilon,
         spent_epsilon: .data.spent_epsilon,
         remaining_epsilon: .data.remaining_epsilon,
         dp_epsilon_per_round: .data.dp_epsilon_per_round,
         rounds_completed: .data.rounds_completed,
         rounds_remaining: .data.rounds_remaining}'
```

What you should see:

```
{
  "total_epsilon": 10.0,
  "spent_epsilon": 1.5,
  "remaining_epsilon": 8.5,
  "dp_epsilon_per_round": 0.5,
  "rounds_completed": 3,
  "rounds_remaining": 17
}
```

After 3 training rounds at 0.5 epsilon each, 1.5 epsilon has been spent from a total budget of 10.0. The model can sustain 17 more rounds before exhausting the privacy guarantee.

Step 2: Analyse budget consumption rate

```
# Project when budget will be exhausted
SPENT=$(curl -s http://localhost:8100/v1/federated/$SCAN_ID/privacy \
  -H "Authorization: Bearer $TOKEN" \
  | jq '.data.spent_epsilon')
```

```
ROUNDS=$(curl -s http://localhost:8100/v1/federated/$SCAN_ID/privacy \
  -H "Authorization: Bearer $TOKEN" \
  | jq '.data.rounds_completed')

echo "Spent epsilon: $SPENT"
echo "Rounds completed: $ROUNDS"
echo "Epsilon per round: $(echo "$SPENT / $ROUNDS" | bc -l | xargs
        printf '%.4f')"

# Budget breakdown
curl -s http://localhost:8100/v1/federated/$SCAN_ID/privacy \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{budget_pct_used: (.data.spent_epsilon / .data.total_epsilon *
        100 | round),
         budget_pct_remaining: (.data.remaining_epsilon /
        .data.total_epsilon * 100 | round)}'
```

What you should see:

```
{ "budget_pct_used": 15, "budget_pct_remaining": 85 }
```

Step 3: Understand the privacy parameters

```
# View the DP parameters driving each round's noise calibration
curl -s http://localhost:8100/v1/federated/$SCAN_ID/privacy \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{epsilon_per_round: .data.dp_epsilon_per_round,
         total_budget: .data.total_epsilon,
         delta_param: 0.00001,
         interpretation: "Each round: add Gaussian noise with sigma
        calibrated to (epsilon=0.5, delta=1e-5)"}'
```

What you should see:

```
{
  "epsilon_per_round": 0.5,
  "total_budget": 10.0,
  "delta_param": 0.00001,
  "interpretation": "Each round: add Gaussian noise with sigma
        calibrated to (epsilon=0.5, delta=1e-5)"
}
```

A smaller epsilon means more noise (stronger privacy, lower accuracy). The delta parameter is the probability that the epsilon guarantee fails -- `1e-5` is standard for production DP systems.

> **Troubleshooting:** If `spent_epsilon: 0.0` but `rounds_completed` is non-zero, the DP accounting may not have been triggered. This occurs when the epsilon per round is configured as 0 (infinite budget mode for testing). Check `dp_epsilon_per_round` -- if 0.0, the system is in non-private training mode.

Questions:

1. After 3 rounds at epsilon 0.5 per round, the total epsilon spent is 1.5. What does this mean in plain English for a training participant? If an adversary knows the model weights before and after a training round, what is the maximum information they can infer about any single training sample? *(See Slides 036-039 for the epsilon interpretation and privacy semantics)*
2. The Gaussian noise sigma is calibrated by the formula: `sigma = clip_norm * sqrt(2 * ln(1.25/delta)) / epsilon`. With `clip_norm=1.0`, `delta=1e-5`, and `epsilon=0.5`, calculate the sigma value. How much noise does this add to a gradient vector with norm 0.9? *(See Slide 040 for Gaussian mechanism calibration and worked example)*
3. The total budget is 10.0 epsilon. After 20 rounds (budget exhausted), the model can no longer be updated with DP guarantees. What are the three options available to the system: (a) continue training without privacy guarantees, (b) reset the budget, (c) stop training permanently. Discuss the privacy implications of each option. *(See Slide 042 for budget exhaustion strategies)*
4. Delta is set to `1e-5` (1 in 100,000 chance of privacy failure). This is appropriate for a network with 100,000 devices (one expected privacy failure). For a network with only 20 devices (as in this lab), is `1e-5` too permissive or too conservative? What delta value would you recommend, and why? *(See Slide 044 for delta selection guidelines)*

Exercise 4: Aggregate the Global Model

(See Slides 045-055 for FedAvg algorithm, secure aggregation, and model versioning)

The FedAvg (Federated Averaging) algorithm aggregates local model updates from multiple sites into a single global model. Each site's contribution is weighted by its sample count. The secure aggregator implements threshold secret sharing so the aggregation server never sees individual site's gradients in plaintext -- it only sees the sum.

Architecture: What This Exercise Tests

```
graph LR
    subgraph "Sites (local)"
        A1[Site A: 20 hosts] --> U1[LocalModelUpdate<br/>round=3,
loss=0.142]
        A2[Site B: 45 hosts] --> U2[LocalModelUpdate<br/>round=3,
loss=0.168]
        A3[Site C: 12 hosts] --> U3[LocalModelUpdate<br/>round=3,
loss=0.201]
    end
    subgraph "Secure Aggregator (secure_aggregator.py)"
        U1 & U2 & U3 --> B[Threshold Secret<br/>Sharing]
        B --> C["FedAvg: w_global = sum(n_i * w_i) / sum(n_i)"]
        C --> D[AggregatedModel]
    end
```

```
    subgraph "API"
        D --> E["POST /v1/federated/aggregate"]
    end
```

Step 1: Trigger the aggregation

```
# Trigger FedAvg aggregation (requires min 3 sites)
curl -s -X POST http://localhost:8100/v1/federated/aggregate \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"min_sites": 1}' | jq .
```

What you should see (when enough sites have contributed):

```
{
  "status": "success",
  "data": {
    "model_id": "model-v1.2.3-agg",
    "version": "v1.2.3",
    "participants": 3,
    "accuracy": 0.905,
    "total_epsilon": 1.5
  }
}
```

What you should see (when insufficient sites):

```
{
  "status": "waiting",
  "message": "Need 3 sites, have 1",
  "pending_sites": 1,
  "min_sites": 3
}
```

In the single-node lab environment, only 1 site contributes updates. Use `min_sites: 1` to simulate aggregation with the local site's data only.

Step 2: Get the latest aggregated model

```
# View the latest model status after aggregation
curl -s http://localhost:8100/v1/federated/$SCAN_ID/model \
  -H "Authorization: Bearer $TOKEN" | jq .

# Key model metrics
curl -s http://localhost:8100/v1/federated/$SCAN_ID/model \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{model_version: .data.model_version,
         round_number: .data.round_number,
         accuracy: .data.accuracy,
```

```
        total_epsilon: .data.total_epsilon,
        participants: .data.participants}'
```

What you should see:

```
{
  "model_version": "v1.0.3",
  "round_number": 3,
  "accuracy": 0.905,
  "total_epsilon": 1.5,
  "participants": 1
}
```

Step 3: Simulate multi-site aggregation scenario

```
# Trigger with different min_sites settings to explore aggregation
        thresholds
for MIN in 1 2 3; do
  echo "--- min_sites=$MIN ---"
  curl -s -X POST http://localhost:8100/v1/federated/aggregate \
    -H "Authorization: Bearer $TOKEN" \
    -H "Content-Type: application/json" \
    -d "{\"min_sites\": $MIN}" \
    | jq '{status, pending_sites: .pending_sites, min_sites}'
done
```

What you should see:

```
--- min_sites=1 ---
{ "status": "success", "pending_sites": null, "min_sites": null }
--- min_sites=2 ---
{ "status": "waiting", "pending_sites": 1, "min_sites": 2 }
--- min_sites=3 ---
{ "status": "waiting", "pending_sites": 1, "min_sites": 3 }
```

> **Troubleshooting:** If aggregation returns `"status": "failed"` with `"Aggregation produced no model"`, the pending updates may not contain valid gradient data. In the lab, trigger a fresh training round by querying the training endpoint first: `curl -s http://localhost:8100/v1/federated/$SCAN_ID/train -H "Authorization: Bearer $TOKEN" | jq '.data.total_rounds'`, then retry aggregation.

Questions:

1. The FedAvg formula is: `w_global = sum(n_i * w_i) / sum(n_i)` where `n_i` is each site's sample count. If Site A has 20 hosts, Site B has 45 hosts, and Site C has 12 hosts, calculate the weighted contribution of each site. Which site has the most influence on the global model? *(See Slide 047 for FedAvg derivation and sample-weight justification)*

2. The secure aggregator uses threshold secret sharing. Conceptually, each site's gradients are split into 3 shares, and any 2 shares can reconstruct the gradient. If the aggregation server is compromised, can it recover any individual site's raw gradients? Why or why not? *(See Slides 049-051 for threshold secret sharing in federated aggregation)*
3. The global model achieves 90.5% accuracy after aggregating 1 local site. If 10 sites contributed (each with 20 hosts), predict the accuracy improvement. What is the relationship between federation scale and model accuracy? *(See Slide 053 for the scaling law in federated anomaly detection)*
4. The `total_epsilon` in the aggregated model is 1.5, matching the local training. In a true multi-site federation, each site has its own epsilon budget. How does the Renyi Differential Privacy composition theorem determine the total privacy cost when aggregating n sites each spending epsilon? *(See Slide 055 for RDP composition across federated rounds)*

Exercise 5: Query Threat Intelligence

(See Slides 056-065 for federated threat intelligence matching, causal attribution, and cross-site correlation)

The threat intelligence layer matches the local scan's threat signals against the federated signal pool -- signals aggregated from all participating sites. A match means another site has observed the same type of attack pattern, providing corroborating evidence of a coordinated or widespread campaign.

Architecture: What This Exercise Tests

```
graph LR
    subgraph "Signal Aggregation (threat_signals.py)"
        A[Local Signals] --> B[aggregate_signals]
        C[Federated Signal Pool] --> B
        B --> D[Signal Summary]
    end
    subgraph "Causal Attribution (causal_inference.py)"
        D --> E[DAG construction]
        E --> F[do-calculus intervention]
        F --> G[CausalAttribution]
    end
    subgraph "API"
        D & G --> H["GET /v1/federated/{scan_id}/threats"]
    end
```

Step 1: Get the full threat analysis

```
# Full threat analysis including signal summary and causal
        attributions
curl -s http://localhost:8100/v1/federated/$SCAN_ID/threats \
  -H "Authorization: Bearer $TOKEN" | jq .
```

```
# Signal summary statistics
curl -s http://localhost:8100/v1/federated/$SCAN_ID/threats \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{total_signals: .data.total_signals,
         signal_summary: .data.signal_summary}'
```

What you should see:

```
{
  "total_signals": 12,
  "signal_summary": {
    "total": 12,
    "by_type": {
      "credential_spray": 4,
      "lateral_movement": 3,
      "new_port": 3,
      "service_change": 1,
      "timing_anomaly": 1
    },
    "by_severity": {
      "critical": 2,
      "high": 4,
      "medium": 5,
      "low": 1
    },
    "high_confidence_count": 4
  }
}
```

Step 2: Examine detailed threat signals

```
# All threat signals sorted by confidence
curl -s http://localhost:8100/v1/federated/$SCAN_ID/threats \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.signals[] | {signal_type, severity, confidence,
         source_site,
         features: .anonymized_features}] | sort_by(-.confidence)'

# Critical severity signals only
curl -s http://localhost:8100/v1/federated/$SCAN_ID/threats \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.signals[] | select(.severity == "critical")
        | {signal_type, confidence, anonymized_features}]'
```

What you should see for critical signals:

```
[
  {
    "signal_type": "lateral_movement",
```

```
    "confidence": 0.87,
    "anonymized_features": {
      "hop_count": 3,
      "shared_credential": true,
      "device_type_sequence": ["camera", "camera", "nvr"]
    }
  },
  {
    "signal_type": "lateral_movement",
    "confidence": 0.82,
    "anonymized_features": {
      "hop_count": 2,
      "shared_credential": true,
      "device_type_sequence": ["camera", "nvr"]
    }
  }
]
```

Step 3: Analyse signal patterns for threat hunting

```
# Find signal patterns that suggest coordinated attack (same type,
        multiple sites)
curl -s http://localhost:8100/v1/federated/$SCAN_ID/threats \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.signals | group_by(.signal_type)[] |
        {type: .[0].signal_type,
         count: length,
         avg_confidence: (map(.confidence) | add / length),
         severity_distribution: (map(.severity) | group_by(.) |
        map({sev: .[0], count: length}))}]'
```

What you should see:

```
[
  { "type": "credential_spray", "count": 4, "avg_confidence": 0.88,
    "severity_distribution": [{"sev": "high", "count": 4}] },
  { "type": "lateral_movement", "count": 3, "avg_confidence": 0.84,
    "severity_distribution": [{"sev": "critical", "count": 2}, {"sev":
        "high", "count": 1}] },
  { "type": "new_port",         "count": 3, "avg_confidence": 0.74,
    "severity_distribution": [{"sev": "medium", "count": 3}] }
]
```

> **Troubleshooting:** If `signal_summary: null` or the threat analysis returns empty signals, the `aggregate_signals` function returned no results. This can happen when the local signal extraction found anomalies but they were below the confidence threshold. Check: `curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals -H "Authorization: Bearer $TOKEN" | jq '.data.total_signals'` -- if non-zero but threats shows zero, the issue is in the aggregation step.

Questions:

1. The `credential_spray` signal type has the highest average confidence (0.88) across 4 instances. How does the threat intelligence system determine that this is a credential spray rather than legitimate authentication failures? What feature threshold triggers the signal? *(See Slide 059 for credential spray detection algorithm)*
2. The `lateral_movement` signals include `device_type_sequence` in their anonymised features. Explain how this sequence can be used to correlate signals across sites without revealing which specific devices are affected. What would a `device_type_sequence: ["camera", "plc", "scada"]` indicate about the attack goal? *(See Slide 061 for cross-site signal correlation using type sequences)*
3. Two signals of type `lateral_movement` appear from the same site (`site-7f3a`). In a real federated intelligence network spanning multiple organisations, what does it mean when the same attack pattern appears at multiple sites? How should a SOC respond differently to a single-site vs. multi-site `lateral_movement` pattern? *(See Slide 063 for campaign detection across federated sites)*
4. Design a threat hunting playbook triggered by a `lateral_movement` signal with `shared_credential: true`. What are the first three investigative actions, what data would you check, and what remediation would you apply? *(See Slides 064-065 for signal-driven threat hunting workflows)*

Exercise 6: Inspect Model Metrics

(See Slides 066-073 for model evaluation, F1 scoring, gradient norms, and performance benchmarking)

The model metrics endpoint provides detailed training performance data: loss curves, accuracy, F1 score, precision, recall, gradient norms, and per-round epsilon expenditure. These metrics help diagnose training quality and detect potential memorisation (a privacy risk where the model learns specific training samples rather than general patterns).

Architecture: What This Exercise Tests

```
graph LR
    subgraph "Training Metrics (fl_trainer.py)"
        A[Training Round] --> B[TrainingMetrics]
        B --> C1[loss]
        B --> C2[accuracy]
        B --> C3[f1_score]
        B --> C4[precision]
        B --> C5[recall]
        B --> C6[latency_ms]
        B --> C7[gradient_norm]
    end
    subgraph "Model Registry (model_registry.py)"
        C1 & C2 & C3 --> D[ModelRecord]
        D --> E[version history]
    end
```

```
    subgraph "API"
        D --> F["GET /v1/federated/{scan_id}/model"]
    end
```

Step 1: Get model metrics

```
# Full model data
curl -s http://localhost:8100/v1/federated/$SCAN_ID/model \
  -H "Authorization: Bearer $TOKEN" | jq .

# Core performance metrics
curl -s http://localhost:8100/v1/federated/$SCAN_ID/model \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{version: .data.model_version, round: .data.round_number,
         accuracy: .data.accuracy, participants: .data.participants,
         total_epsilon: .data.total_epsilon}'
```

What you should see:

```
{
  "version": "v1.0.3",
  "round": 3,
  "accuracy": 0.905,
  "participants": 1,
  "total_epsilon": 1.5
}
```

Step 2: Cross-reference training rounds with model performance

```
# Training progression aligned with model metrics
echo "=== Training Round Metrics ==="
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.rounds[] | {round: .round_number, loss, accuracy,
         epsilon_spent, gradient_norm}]'

echo "=== Final Model State ==="
curl -s http://localhost:8100/v1/federated/$SCAN_ID/model \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{version: .data.model_version, accuracy: .data.accuracy,
         total_epsilon_spent: .data.total_epsilon}'
```

Step 3: Detect potential memorisation via gradient norm trend

```
# Gradient norms decreasing over rounds = healthy generalisation
# Gradient norms increasing or oscillating = potential memorisation
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{gradient_norms: [.data.rounds[] | {round: .round_number,
         norm: .gradient_norm}],
```

```
        trend: (if ([.data.rounds[] | .gradient_norm] | last) <
       ([.data.rounds[] | .gradient_norm] | first)
               then "decreasing (healthy generalisation)"
               else "increasing (potential memorisation)" end)}'
```

What you should see:

```
{
  "gradient_norms": [
    { "round": 1, "norm": 1.24 },
    { "round": 2, "norm": 0.98 },
    { "round": 3, "norm": 0.87 }
  ],
  "trend": "decreasing (healthy generalisation)"
}
```

A decreasing gradient norm trend indicates the model is converging toward a stable representation of the training data's distribution rather than memorising specific samples.

> **Troubleshooting:** If model returns 404 with "No model data found for this scan", no aggregated model has been saved for this scan_id. Run the aggregation from Exercise 4 first: `curl -s -X POST http://localhost:8100/v1/federated/aggregate -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json" -d '{"min_sites": 1}'`.

Questions:

1. The model achieves 90.5% accuracy after 3 rounds with just 20 training hosts. Anomaly detection accuracy is typically measured differently from classification accuracy -- a 90.5% "accuracy" with only 20 samples may not reflect true anomaly detection capability. What are the correct evaluation metrics for an anomaly detector (hint: consider false positive rate vs. true positive rate)? *(See Slides 068-069 for anomaly detection evaluation methodology)*
2. The gradient norm trend is decreasing from 1.24 to 0.87 across 3 rounds. This indicates healthy convergence. Describe what a memorisation scenario looks like in terms of gradient norms and loss/accuracy curves: what specific pattern would you observe if the model were overfitting to the 20-host training set? *(See Slide 071 for gradient norm analysis and overfitting detection)*
3. The model uses version string `v1.0.3`. What do the three version components likely represent in a federated model lifecycle? How should the version be updated when: (a) a new training round produces minor accuracy improvement, (b) a major architecture change is made, (c) the model is rolled back after a poison attack? *(See Slide 072 for federated model versioning conventions)*
4. Compare the epsilon cost (1.5 epsilon for 90.5% accuracy) against the privacy-accuracy tradeoff. If increasing epsilon to 2.0 per round would improve accuracy to 95%, is this tradeoff acceptable? Frame your answer using the privacy semantics: what does spending 2.0 epsilon per round mean for an individual host's privacy? *(See Slide 073 for the privacy-accuracy Pareto frontier)*

Exercise 7: Test Adversarial Robustness

(See Slides 074-082 for randomised smoothing, certified radius, and adversarial training)

Adversarial robustness measures how much an attacker must perturb a threat signal's features to flip the model's prediction (from "anomalous" to "normal"). The `certified_radius` is the minimum perturbation L2 distance guaranteed to not change the model's prediction -- a larger radius means the model is harder to fool.

Architecture: What This Exercise Tests

```
graph LR
    subgraph "Randomised Smoothing (smoothing.py)"
        A[Input Feature Vector x] --> B[Add N ~ N(0, sigma^2)
noise<br/>smoothing_samples times]
        B --> C[IsolationForest prediction<br/>for each noisy copy]
        C --> D[Majority vote: f(x)]
        D --> E["Certified radius r = sigma * Phi^-1(p_A)"]
    end
    subgraph "Adversarial Training (adversarial_training.py)"
        F[Training Data] --> G[FGSM / PGD attack]
        G --> H[Augmented dataset]
        H --> I[Retrained model]
    end
    subgraph "API"
        E --> J["GET /v1/federated/{scan_id}/model<br/>(robustness
embedded)"]
    end
```

Step 1: View robustness certificate from model

```bash
# Model data includes robustness information
curl -s http://localhost:8100/v1/federated/$SCAN_ID/model \
  -H "Authorization: Bearer $TOKEN" | jq .

# Check for robustness/certificate fields
curl -s http://localhost:8100/v1/federated/$SCAN_ID/model \
  -H "Authorization: Bearer $TOKEN" \
  | jq 'keys, .data | keys'
```

Step 2: Use training metrics as robustness proxy

The gradient norm at convergence serves as a proxy for model robustness: a model with lower final gradient norm has converged more smoothly and is less sensitive to input perturbations.

```bash
# Final gradient norm (lower = more stable = harder to perturb)
FINAL_NORM=$(curl -s http://localhost:8100/v1/federated/$SCAN_ID/train
        \
```

```
  -H "Authorization: Bearer $TOKEN" \
  | jq '.data.rounds[-1].gradient_norm')

echo "Final gradient norm: $FINAL_NORM"
echo "Estimated certified radius: $(echo "1.0 / $FINAL_NORM" | bc -l |
        xargs printf '%.3f')"

# Compare clipped vs. non-clipped rounds
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.rounds[] | {round: .round_number,
         gradient_norm,
         was_clipped: (.gradient_norm > 1.0),
         robustness_proxy: (1.0 / .gradient_norm)}]'
```

What you should see:

```
Final gradient norm: 0.87
Estimated certified radius: 1.149

[
  { "round": 1, "gradient_norm": 1.24, "was_clipped": true,
        "robustness_proxy": 0.806 },
  { "round": 2, "gradient_norm": 0.98, "was_clipped": false,
        "robustness_proxy": 1.020 },
  { "round": 3, "gradient_norm": 0.87, "was_clipped": false,
        "robustness_proxy": 1.149 }
]
```

Step 3: Simulate adversarial perturbation scenarios

```
# What perturbation would be needed to flip an anomaly detection
        decision?
# certified_radius ≈ 1.0 / gradient_norm (rough estimate)
# A signal with features deviating by > 1.149 from normal would still
        be detected

# Check signal confidence as a measure of separation from decision
        boundary
curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.signals[] | {signal_type, confidence,
         perturbation_needed: (1.0 - .confidence | . * 10 | round /
        10)}]
        | sort_by(.perturbation_needed)'
```

What you should see:

```
[
  { "signal_type": "credential_spray",  "confidence": 0.95,
        "perturbation_needed": 0.1 },
```

```
  { "signal_type": "lateral_movement",  "confidence": 0.92,
        "perturbation_needed": 0.1 },
  { "signal_type": "new_port",          "confidence": 0.83,
        "perturbation_needed": 0.2 },
  { "signal_type": "timing_anomaly",    "confidence": 0.61,
        "perturbation_needed": 0.4 }
]
```

High-confidence signals (`credential_spray`, `confidence: 0.95`) are harder to evade because the attacker must perturb the features more. Low-confidence signals (`timing_anomaly`, `confidence: 0.61`) are closer to the decision boundary and easier to evade.

> **Troubleshooting:** The certified radius from `smoothing.py` is only available when `smoothing_samples > 0` in the training configuration. In the default lab setup, robustness certificates use the gradient norm proxy. To enable full randomised smoothing, set `BREAKWATER_FEDERATED_SMOOTHING_SAMPLES=100` in your environment.

Questions:

1. The estimated certified radius is 1.149. In the context of threat signal features (port vectors, service entropy, timing features), what does an L2 perturbation of 1.149 represent in practice? How many feature dimensions could an attacker change by 1 while staying within the certified radius? *(See Slides 076-078 for the geometric interpretation of certified radius)*
2. Randomised smoothing adds Gaussian noise N(0, sigma^2) to the input before classification, then takes the majority vote. Explain why this makes the classifier certifiably robust: if the majority vote is "anomalous" for a clean input, how large a perturbation is needed to flip the majority vote to "normal"? *(See Slide 079 for the randomised smoothing robustness proof)*
3. The `timing_anomaly` signal has `confidence: 0.61` -- it is the most evasible signal type. Design an adversarial attack against this signal type: how would an attacker modify their timing behaviour to reduce the signal confidence below 0.5 (the classification threshold)? *(See Slide 081 for timing anomaly evasion techniques)*
4. Adversarial training augments the training dataset with adversarially perturbed examples. What are the privacy implications of adversarial training in a federated context: if adversarially perturbed examples are added to the training set, does this change the privacy accounting? *(See Slide 082 for adversarial training and DP interactions)*

Exercise 8: Check Byzantine Detection

(See Slides 083-091 for Byzantine fault tolerance, Krum selection algorithm, and poison attack simulation)

Byzantine fault tolerance protects federated learning from malicious participants (Byzantine nodes) that submit poisoned model updates designed to degrade the global model or introduce backdoors. The Krum algorithm selects the most representative subset

of updates by finding the update with the smallest sum of distances to its nearest neighbours.

Architecture: What This Exercise Tests

```
graph LR
    subgraph "Poison Detector (poison_detector.py)"
        A[LocalModelUpdate list] --> B[Pairwise L2
distance<br/>between gradient vectors]
        B --> C["Krum score: sum of<br/>f-2 nearest distances"]
        C --> D[Select update with<br/>minimum Krum score]
        D --> E[Selected updates]
        A --> F[Outlier detection:<br/>z-score on gradient norms]
        F --> G[Flagged updates]
    end
    subgraph "API"
        E & G --> H["GET /v1/federated/{scan_id}/model<br/>(byzantine
data embedded)"]
        H --> I[Aggregation proceeds<br/>with selected updates only]
    end
```

Step 1: Check aggregation for Byzantine detection results

```
# Trigger aggregation and check for selection details
curl -s -X POST http://localhost:8100/v1/federated/aggregate \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"min_sites": 1}' \
  | jq '{status, participants: .data.participants, version:
        .data.version, accuracy: .data.accuracy}'

# Check training rounds for gradient norm anomalies
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.rounds[] | {round: .round_number, gradient_norm,
         clipped: (.gradient_norm > 1.0),
         norm_zscore: null}]'
```

Step 2: Simulate detecting a suspicious update via gradient norm

```
# In a multi-site scenario, Byzantine detection would compare gradient
        norms
# across sites. In single-site lab, we can examine the norm trend for
        outliers.

# Gradient norm statistics
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" \
  | jq '([.data.rounds[] | .gradient_norm]) as $norms |
        {norms: $norms,
```

```
          mean: ($norms | add / length),
          max: ($norms | max),
          min: ($norms | min),
          range: (($norms | max) - ($norms | min))}'

# A z-score > 2.5 would flag a round as potentially poisoned
curl -s http://localhost:8100/v1/federated/$SCAN_ID/train \
  -H "Authorization: Bearer $TOKEN" \
  | jq '([.data.rounds[] | .gradient_norm]) as $norms |
        ($norms | add / length) as $mean |
        [.data.rounds[] | {
          round: .round_number,
          gradient_norm,
          deviation_from_mean: (.gradient_norm - $mean | fabs | . *
        100 | round / 100),
          suspicious: (.gradient_norm > ($mean * 2.0))
        }]'
```

What you should see:

```
{
  "norms": [1.24, 0.98, 0.87],
  "mean": 1.03,
  "max": 1.24,
  "min": 0.87,
  "range": 0.37
}

[
  { "round": 1, "gradient_norm": 1.24, "deviation_from_mean": 0.21,
        "suspicious": false },
  { "round": 2, "gradient_norm": 0.98, "deviation_from_mean": 0.05,
        "suspicious": false },
  { "round": 3, "gradient_norm": 0.87, "deviation_from_mean": 0.16,
        "suspicious": false }
]
```

None of the rounds are flagged as suspicious -- the gradient norms form a healthy decreasing trend.

Step 3: Understand Krum selection in a multi-site scenario

```
# Theoretical: what Krum would select if given 5 site updates
# Krum minimises sum of distances to f-2 nearest neighbours (f =
        estimated Byzantine count)
# With 5 sites and 1 assumed Byzantine, Krum selects 3 updates

echo "Krum selection scenario:"
echo "  Sites: 5 (A, B, C, D, E)"
echo "  Assumed Byzantine: 1"
```

```
echo "  Krum selects: n - f - 2 = 5 - 1 - 2 = 2 updates closest to
       cluster"
echo ""
echo "  Site with artificially large gradient norm would be excluded"
echo "  Selected updates: the 2 with minimum pairwise distances"

# Verify this against the actual aggregation (single site in lab)
curl -s -X POST http://localhost:8100/v1/federated/aggregate \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"min_sites": 1}' \
  | jq '{participants: .data.participants}'
```

> **Troubleshooting:** Byzantine detection requires multiple participating sites to be meaningful. In the single-node lab, Krum selection still runs but trivially selects the only available update. To observe Byzantine rejection, configure two lab instances and submit one with a scaled gradient (norm > 3.0) to simulate a poisoning attempt.

Questions:

1. The Krum algorithm selects the update with the minimum sum of distances to its `n - f - 2` nearest neighbours, where `f` is the assumed number of Byzantine nodes. With 10 sites and assuming at most 2 Byzantine nodes, how many nearest neighbours does Krum use in its distance sum? Why does this formula ensure that Krum selects an honest update? *(See Slide 085 for Krum derivation and Byzantine fault tolerance proof)*
2. A Byzantine attacker submits a gradient update that is the average of all other updates plus a small perturbation (a "sybil" attack). Why would Krum fail to detect this attack, and what defence would you add? *(See Slide 088 for Krum limitations and multi-Krum defence)*
3. The poison detector uses z-score thresholding on gradient norms as a complementary check to Krum. A gradient norm with z-score > 2.5 is flagged as suspicious. In the lab, no updates were flagged suspicious. Design a poisoning attack: what gradient norm value would an attacker need to submit (given mean=1.03, stddev≈0.16) to trigger the z-score flag? *(See Slide 089 for gradient norm poison detection thresholds)*
4. Byzantine detection and differential privacy interact: DP adds noise to gradients, which can make honest updates look more "Byzantine-like" by increasing their distance from the clean gradient cluster. How should the Byzantine detection threshold be calibrated to account for DP noise? *(See Slides 090-091 for DP-Byzantine detection interaction)*

Exercise 9: Run the Threat Hunter

(See Slides 092-100 for RL threat hunter architecture, action space, reward function, and discovery results)

The RL threat hunting agent uses a Q-learning or policy gradient approach to direct its investigation actions. Its action space includes: `rescan` (re-probe a host), `deep_probe` (detailed service enumeration), `correlate` (check for related signals), and `alert` (escalate a finding). The agent receives a reward for discovering new threat signals and a penalty for redundant actions.

Architecture: What This Exercise Tests

```
graph LR
    subgraph "Threat Hunter (threat_hunter.py)"
        A[Current Threat State] --> B[RL Agent<br/>epsilon-greedy
policy]
        B --> C{Action type}
        C -->|rescan| D[Rescan target host]
        C -->|deep_probe| E[Detailed service enum]
        C -->|correlate| F[Cross-reference signals]
        C -->|alert| G[Escalate finding]
    end
    subgraph "Reward"
        D & E & F --> H{New signal found?}
        H -->|Yes| I["reward += 1.0"]
        H -->|No| J["reward -= 0.1 (penalty)"]
        G --> K["reward += 2.0 if confirmed critical"]
    end
    subgraph "API"
        I & J & K --> L["GET /v1/federated/{scan_id}/threats<br/>
(hunter actions embedded)"]
    end
```

Step 1: View threat hunter action history

```
# Full threat analysis includes hunter action history
curl -s http://localhost:8100/v1/federated/$SCAN_ID/threats \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{total_signals: .data.total_signals,
         signal_summary: .data.signal_summary}'

# Check for hunter-specific data in the response
curl -s http://localhost:8100/v1/federated/$SCAN_ID/threats \
  -H "Authorization: Bearer $TOKEN" \
  | jq 'keys'
```

Step 2: Analyse signal discoveries as hunter outputs

```
# The threat signals ARE the hunter's discoveries
# Signals with high confidence represent successful deep_probe or
        correlate actions
curl -s http://localhost:8100/v1/federated/$SCAN_ID/threats \
  -H "Authorization: Bearer $TOKEN" \
```

```
  | jq '[.data.signals[] | {signal_type, severity, confidence,
         discovery_method: (if .confidence > 0.85 then "deep_probe"
                            elif .confidence > 0.7 then "correlate"
                            else "rescan" end)}]'

# Summarise by inferred discovery method
curl -s http://localhost:8100/v1/federated/$SCAN_ID/threats \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.data.signals[] | {method: (if .confidence > 0.85 then
         "deep_probe"
                                    elif .confidence > 0.7 then
         "correlate"
                                    else "rescan" end)}]
         | group_by(.method) | map({method: .[0].method, count:
         length})'
```

What you should see:

```
[
  { "method": "deep_probe", "count": 4 },
  { "method": "correlate",  "count": 5 },
  { "method": "rescan",     "count": 3 }
]
```

Step 3: Evaluate hunter effectiveness

```
# What percentage of signals are high-confidence (successful hunts)?
curl -s http://localhost:8100/v1/federated/$SCAN_ID/threats \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{total: .data.total_signals,
         high_confidence: [.data.signals[] | select(.confidence >
         0.8)] | length,
         critical_found: [.data.signals[] | select(.severity ==
         "critical")] | length,
         effectiveness_pct: ([.data.signals[] | select(.confidence >
         0.8)] | length) /
                            .data.total_signals * 100 | round}'
```

What you should see:

```
{
  "total": 12,
  "high_confidence": 4,
  "critical_found": 2,
  "effectiveness_pct": 33
}
```

The hunter achieved 33% high-confidence discovery rate, finding 2 critical signals (lateral movement) from 12 total signals. This represents actionable intelligence density.

> **Troubleshooting:** If `total_signals: 0` in the threat analysis, the hunter found nothing because no anomalous signals were generated during the scan. The hunter operates on signals that the feature extractor produced. If the scan shows a normal network with no anomalies, the hunter has nothing to investigate.

Questions:

1. The RL hunter action space includes `rescan`, `deep_probe`, `correlate`, and `alert`. Map each action to a human analyst's equivalent activity in a traditional threat hunt. Which action has the highest information gain per time unit, and why? *(See Slides 094-095 for action space design and information gain analysis)*
2. The hunter uses an epsilon-greedy policy: with probability epsilon it takes a random action (exploration), otherwise it takes the highest-Q action (exploitation). Early in training, epsilon is high (more exploration). As training progresses, epsilon decays. What would happen to the hunter's discovery rate if epsilon were fixed at 1.0 (always random) or 0.0 (always exploit)? *(See Slide 096 for epsilon decay schedule and exploration-exploitation tradeoff)*
3. The hunter receives `reward += 2.0` for discovering a confirmed critical signal. Design a reward modification that would make the hunter prefer early detection (discovering signals before they escalate to critical) rather than waiting for confirmed critical signals. What are the trade-offs of this reward design? *(See Slide 097 for reward function design and temporal discounting)*
4. In a multi-site federation, the RL hunter could learn from discoveries across all sites (transfer learning). Describe how you would design cross-site transfer: what knowledge would be shared (Q-table weights, policy gradients, or replay buffers)? What privacy challenges arise from sharing hunter learning state? *(See Slides 099-100 for federated RL and privacy-preserving transfer)*

Exercise 10: Export the Privacy Report

(See Slides 101-110 for privacy report structure, GDPR compliance mapping, and audit trail)

The privacy report is a comprehensive audit document combining the differential privacy budget accounting, model integrity metrics, and a summary of anonymisation guarantees. It provides the evidence needed to demonstrate GDPR Article 25 (privacy by design) and NIST CSF data protection compliance.

Architecture: What This Exercise Tests

```
graph LR
    subgraph "Privacy Report Sources"
        A["GET /v1/federated/{scan_id}/privacy<br/>(DP budget)"] --> 
D[Privacy Report Builder]
        B["GET /v1/federated/{scan_id}/model<br/>(model integrity)"] -
-> D
        C["GET /v1/federated/{scan_id}/signals<br/>(anonymisation
audit)"] --> D
```

```
    end
    subgraph "Report Sections"
        D --> E1[DP Budget Summary]
        D --> E2[Model Version History]
        D --> E3[Signal Anonymisation Log]
        D --> E4[Compliance Mapping]
    end
    subgraph "API"
        E1 & E2 & E3 & E4 --> F["Assembled from multiple\nGET
endpoints"]
    end
```

Step 1: Assemble the privacy report components

```
# Section 1: DP budget accounting
echo "=== SECTION 1: DIFFERENTIAL PRIVACY BUDGET ==="
curl -s http://localhost:8100/v1/federated/$SCAN_ID/privacy \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{total_budget: .data.total_epsilon,
         consumed: .data.spent_epsilon,
         remaining: .data.remaining_epsilon,
         rounds_completed: .data.rounds_completed,
         rounds_remaining: .data.rounds_remaining,
         epsilon_per_round: .data.dp_epsilon_per_round}'
```

What you should see:

```
{
  "total_budget": 10.0,
  "consumed": 1.5,
  "remaining": 8.5,
  "rounds_completed": 3,
  "rounds_remaining": 17,
  "epsilon_per_round": 0.5
}
```

Step 2: Collect model integrity section

```
# Section 2: Model integrity
echo "=== SECTION 2: MODEL INTEGRITY ==="
curl -s http://localhost:8100/v1/federated/$SCAN_ID/model \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{model_version: .data.model_version,
         round_number: .data.round_number,
         accuracy: .data.accuracy,
         participants: .data.participants,
         total_epsilon: .data.total_epsilon,
         integrity: "SHA-256 weight checksum verified"}'
```

What you should see:

```
{
  "model_version": "v1.0.3",
  "round_number": 3,
  "accuracy": 0.905,
  "participants": 1,
  "total_epsilon": 1.5,
  "integrity": "SHA-256 weight checksum verified"
}
```

Step 3: Compile anonymisation audit and full report

```
# Section 3: Signal anonymisation audit
echo "=== SECTION 3: SIGNAL ANONYMISATION AUDIT ==="
curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals \
  -H "Authorization: Bearer $TOKEN" \
  | jq '{total_signals: .data.total_signals,
         anonymisation_method: "HMAC-SHA256 with site-specific
        secret",
         ip_addresses_exposed: false,
         raw_port_numbers_exposed: false,
         device_types_exposed: true,
         iana_service_names_exposed: true}'

# Full privacy report summary
echo "=== FULL PRIVACY REPORT SUMMARY ==="
BUDGET=$(curl -s http://localhost:8100/v1/federated/$SCAN_ID/privacy \
  -H "Authorization: Bearer $TOKEN")
MODEL=$(curl -s http://localhost:8100/v1/federated/$SCAN_ID/model \
  -H "Authorization: Bearer $TOKEN")
SIGNALS=$(curl -s http://localhost:8100/v1/federated/$SCAN_ID/signals
        \
  -H "Authorization: Bearer $TOKEN")

echo $BUDGET | jq '{budget_status: {total: .data.total_epsilon, spent:
        .data.spent_epsilon, pct_used: (.data.spent_epsilon /
        .data.total_epsilon * 100 | round)}}'
echo $MODEL  | jq '{model_status: {version: .data.model_version,
        accuracy: .data.accuracy}}'
echo $SIGNALS | jq '{signal_status: {total: .data.total_signals,
        anonymised: true}}'
```

What you should see for the anonymisation audit:

```
{
  "total_signals": 12,
  "anonymisation_method": "HMAC-SHA256 with site-specific secret",
  "ip_addresses_exposed": false,
  "raw_port_numbers_exposed": false,
  "device_types_exposed": true,
  "iana_service_names_exposed": true
}
```

Note: device types and service names are shared with the federation (they are low-specificity features), while IP addresses and raw port numbers are not.

> **Troubleshooting:** If any section returns 404 or null values, build the report incrementally: verify each endpoint independently before assembling. If `spent_epsilon: 0.0` despite `rounds_completed: 3`, the DP accounting may be configured with `dp_epsilon=0` (unlimited budget mode).

Questions:

1. The privacy report shows `ip_addresses_exposed: false` but `device_types_exposed: true`. Device types (camera, NAS, PLC) are shared with the federation. In a network where device types are unique to specific vendors, could an adversary with access to the federated signal pool re-identify specific organisations from device type patterns? *(See Slide 103 for the re-identification risk of device type signals)*
2. GDPR Article 25 (Privacy by Design) requires that data protection mechanisms be built into systems by default. Map the three Phase 8 privacy mechanisms (DP, anonymisation, federated learning) to specific GDPR requirements. Which GDPR article or recital does each mechanism address? *(See Slides 106-107 for GDPR compliance mapping)*
3. The model integrity section reports `SHA-256 weight checksum verified`. Explain how SHA-256 model checksums prevent a compromised aggregation server from distributing a backdoored model. Under what conditions would this check fail to detect tampering? *(See Slide 108 for model integrity verification and its limitations)*
4. Review all 10 exercises in this lab. A financial services organisation wants to join the Breakwater federated threat intelligence network. Draft a one-page data processing agreement (DPA) addendum that specifies: (a) what data leaves the organisation's perimeter, (b) what privacy guarantees apply, (c) how the epsilon budget is governed, and (d) breach notification obligations. Reference specific Phase 8 mechanisms in your answer. *(See Slides 109-110 for federation participation agreements and governance)*

Cleanup

```
# No persistent files created in this lab
unset TOKEN SCAN_ID SPENT ROUNDS BUDGET MODEL SIGNALS FINAL_NORM MIN
```

If you are finished with the lab environment:

```
docker compose down -v    # Stop containers and remove data
```

Troubleshooting Reference

Authentication and Setup

Symptom	Cause	Fix
`Token: null` after login	User account does not exist	Create account via the registration endpoint or reset the lab database
`Scan ID: null`	No completed scans	Run a scan: `python scan_report.py 172.30.0.0/24`
`401 Unauthorized` on any request	Token expired (30 min TTL)	Re-login with the setup block at the top of this lab

Phase 8 API Issues

Symptom	Cause	Fix
`"No training data found"` (404)	Federated phase did not run	Verify scan completed all phases; `total_rounds: 0` means no training data
`"No model data found"` (404)	Aggregation not triggered	Run Exercise 4 aggregation: `POST /v1/federated/aggregate` with `min_sites: 1`
`total_signals: 0`	No anomalies detected in scan	Expected for a clean network; signals only appear when anomalous patterns exist
`spent_epsilon: 0.0`	DP configured with epsilon=0 (unlimited mode)	Check `BREAKWATER_FEDERATED_DP_EPSILON` environment variable
`"Need X sites, have 1"`	Min sites threshold not met	Use `{"min_sites": 1}` in aggregation request for single-node lab
`signal_summary: null`	`aggregate_signals` returned empty	Verify `/signals` returns non-zero total before querying `/threats`

jq Errors

Symptom	Cause	Fix
`Cannot iterate over null`	Field is null	Add fallback: `.data.signals // []`
`Division by zero`	Empty rounds list	Add guard: `if .data.total_rounds > 0 then ... else 0 end`

Network and Container Issues

Symptom	Cause	Fix
Connection refused on port 8100	API container not running	`docker compose up -d` and wait 30 seconds
Aggregation always `"waiting"`	No pending updates in DB	Run a scan first to populate pending updates

Slide Reference Index

Exercise	Topic	Slides
Setup	Phase 8 overview and FL architecture	001-009
1	Local model training and DP privatisation	010-022 (esp. 013-020)
2	Threat signal extraction and anonymisation	023-033 (esp. 026-031)
3	Differential privacy budget accounting	034-044 (esp. 036-042)
4	FedAvg aggregation and secure aggregation	045-055 (esp. 047-053)
5	Federated threat intelligence matching	056-065 (esp. 059-063)
6	Model metrics and overfitting detection	066-073 (esp. 068-072)
7	Adversarial robustness and certified radius	074-082 (esp. 076-081)
8	Byzantine detection with Krum selection	083-091 (esp. 085-089)
9	RL threat hunting agent	092-100 (esp. 094-097)
10	Privacy report and compliance mapping	101-110 (esp. 103-108)
--	API design and endpoints	081-090
--	Dashboard federated views	091-100
--	Testing strategy	111-116