Course Project Milestone 2 — Logging & Mini-SIEM (Week 2)

This week: Turn your Week-1 capture/decoder into a logging-first agent and a tiny local SIEM with basic correlation and CLI views.

---

1) What's new vs Week-1 (changelog)

Clear functional map → code hooks. Every requirement points to a specific file/function to edit.

Structured logs (JSONL) split into detections and ops with `schema_version: "1.0"`.

Mini-SIEM: ingest detections → correlate → write `alerts.jsonl` and print alerts.

Consistent Ctrl-C in both pcap and live loops (graceful stop + log flush). We break out of the packet loop when the stop flag is set.

Visible tests are explicit and runnable with `scripts/run_tests.py` (no guesswork about automation).

HTTP detection is optional (bonus). Required detections: DNS, ARP, ICMP.

---

2) Learning outcomes

By the end of this milestone, you can:

1. Produce structured security event logs in JSONL with stable schema.

2. Implement a mini-SIEM pipeline (ingest → correlate → alert) with simple CLI views.

3. Handle SIGINT (Ctrl-C) gracefully and consistently in both pcap and live modes.

4. Build repeatable visible tests that run locally and in CI.

---

3) Folder Structure (Follow it for the least file path related bugs)

Course Project

- Agent
  - _init_.py
  - capture.py
  - cli.py
  - logging.py
  - signals.py
  - utlis.py
- Detectors
  - _init_.py
  - arp.py
  - dns.py
  - http.py
  - icmp.py
- Siem
  - _init_.py
  - alerts.py
  - ingest.py
  - mini_siem.py
- Scripts
  - run_tests.py
- Pcap
- Logs
- README.md
- Config.yaml
- Requirements.txt (Tools and Version needed to run the project.
- Course Project Milestone 2.docx (this file)

Which files need to have added code (hooks):

- DNS: `detectors/dns.py`

- ICMP: `detectors/icmp.py`

- ARP: `detectors/arp.py`

- (Optional) HTTP: `detectors/http.py`

- SIEM correlation/alerts: `siem/mini_siem.py` + `siem/alerts.py`

- Consistent Ctrl-C handling is already wired in `agent/cli.py` and `agent/signals.py`.

---

4) Event schemas:

4.1 Detections (./logs/detections.jsonl)

Each line is one JSON object:

```
{
  "ts": "ISO8601-with-timezone",
  "schema_version": "1.0",
  "src": "ip[:port]",
  "dst": "ip[:port]",
  "proto": "TCP|UDP|ICMP|ARP|DNS|HTTP|OTHER",
  "rule_id": "STRING",
  "severity": "low|medium|high",
  "summary": "short human-readable sentence",
  "metadata": { "freeform": "small key/values" }
}
```

4.2 Ops log (./logs/ops.jsonl)

```
{
  "ts": "ISO8601",
  "level": "DEBUG|INFO|WARN|ERROR",
  "component": "runner|capture|decoder|detector|siem",
  "msg": "start_pcap|start_live|shutdown|packet_error|...",
  "kv": { "file": "...", "iface": "...", "reason": "..." }
}
```

4.3 Alerts (./logs/alerts.jsonl)

```
{
  "ts": "ISO8601",
  "alert_id": "ALERT_ICMP_FLOOD|ALERT_REPEATED_RULE|...",
  "severity": "low|medium|high",
  "summary": "why it fired",
  "entities": { "src": "1.2.3.4", "rule_id": "..." },
  "evidence_count": 42
}
```

---

5) Required detections (minimum)

Implement these heuristics; keep thresholds in config.yaml.

- DNS_SUSPICIOUS (Required)
  Trigger if any DNS query name has:
  - a label > dns_label_max (default 63 chars), or
  - a total length > dns_name_max (default 253 chars), or
  - high subdomain entropy (e.g., ≥ 4.0 bits/char).
    Log name + entropy in metadata.

- ICMP_RATE (Required)
  Sliding 1-second window; if echo requests from same src exceed icmp_per_sec, emit high-severity detection. Include rate & threshold in metadata.
- ARP_MULTIMAC (Required)
  Within arp_window_sec, if the same IP is seen with ≥ 2 different MACs, emit medium-severity detection. Include the observed MACs and window in metadata.
- HTTP_KEYWORD (Optional / Bonus)
  Simple match of suspicious terms (e.g., "admin", "password", "login") in HTTP host/URI.

Clarification for Week-1: HTTP parsing was only ever a *bonus*. You are not penalized for omitting it in Milestone 2.

---

6) Running the agent & SIEM

Setup:

- python -m venv .venv && source .venv/bin/activate
- pip install -r requirements.txt

PCAP Mode:

- python -m idsips.agent.cli pcap --pcap ./pcaps/dns_examples.pcapng

Live mode (Can be used in Command Line as well):

- python -m idsips.agent.cli live --iface lo --dry-run
- # Press Ctrl-C (SIGINT) to stop – both live and pcap modes break the packet loop on stop.

Mini-SIEM views:

- python -m idsips.siem.mini_siem --timeline
- python -m idsips.siem.mini_siem --top
- python -m idsips.siem.mini_siem --rule-stats

---

7) Visible tests (Meaning & How to run)

Definition: *Visible tests* are the exact, published checks that the grader will run. You can run them locally and in CI with a single script.

Run all tests with the following script:

python ./scripts/run_tests.py

What they check (at minimum):

1. **PCAP DNS** — DNS_SUSPICIOUS entries appear in detections.jsonl; ops shows start & shutdown.
2. **ARP spoof** — if pcaps/arp_spoof_short.pcap is present, detect ARP_MULTIMAC.
3. **Live SIGINT** — --dry-run process exits 0 upon SIGINT; ops shows shutdown.
4. **Mini-SIEM** — CLI runs and (if data supports it) writes alerts.jsonl.

Note on automation: The SIGINT test uses Python's subprocess and proc.send_signal(signal.SIGINT) to **gracefully** stop the live loop—no platform-specific PID hacks required.

---

8) Deliverables (what to submit)

- README.md (how to run agent + SIEM; sample outputs/screens)
- config.yaml (your chosen thresholds)
- agent/, detectors/, siem/, scripts/ (your code)
- pcaps/ (include the two provided; optional ARP pcap if used) (You can also use your own pcaps if you prefer those, just make sure to include those in the final submission)
- A short example logs/detections.jsonl and logs/ops.jsonl created from the provided/individual pcap (not huge files)

Packaging:
Submit a single zip named milestone2_fullname.zip.

---

9) Rubric (100 pts)

Core (80 pts)

- JSONL logging schema (events + ops) implemented exactly (20)
- DNS detection & logging (15)
- ICMP flood heuristic & logging (10)
- ARP spoof heuristic & logging (10)
- Graceful shutdown in both modes (Ctrl-C) with proper ops entries (15)
- Mini-SIEM: ingest, correlation, alerts.jsonl + CLI views (10)

Quality (20 pts)

- Config-driven thresholds; clear README with commands + samples (10)
- Code clarity: TODOs resolved, docstrings, small functions (5)
- scripts/run_tests.py prints clean pass/fail and exits non-zero on failure (5)

Bonus (up to +5)

- Optional HTTP keyword detection with clean metadata (+0–5)

Hidden checks may validate: valid UTF-8 JSONL; no crashes on malformed frames; shutdown logs on SIGINT.

---

10) Troubleshooting

- **No detections written?** Check rules.* toggles and thresholds in config.yaml.
- **Live mode needs root?** Use --dry-run for tests/CI or run with appropriate permissions on your OS.
- **JSON decode errors in SIEM?** Ensure each line in detections.jsonl is a **single JSON object**.
- **Ctrl-C didn't exit?** Verify you're on the live/pcap loop that **breaks** when the SIGINT flag is set (already in the sample code).

IDS/IPS Project – Files Overview & Functionalities (Milestone 2)

This will explain the purpose and main functionality of each file/folder in the sample code. It also notes how capture_logger.py from Part 1 is reused or referenced.

Top-level

- config.yaml – Central configuration. Enables/disables rules, sets thresholds, selects logs directory, toggles live dry-run and optional per-packet ops logging.
- requirements.txt – Python dependencies (PyShark, PyYAML, orjson, rich).
- README.md – Run instructions for agent, mini-SIEM, and visible tests; include sample outputs/screens.
- logs/ – Runtime output: detections.jsonl (events), ops.jsonl (operational), alerts.jsonl (SIEM).
- pcaps/ – Input captures used for development and visible tests (basic_http.pcapng, dns_examples.pcapng). You may add arp_spoof_short.pcap.
- scripts/run_tests.py – Student-visible automated test runner (must pass on grader's machine).

Package: idsips/agent/

- cli.py – CLI entrypoint for the agent (subcommands: pcap, live, SIGINT handling with loop break on stop). TODO anchors show where to add detectors/subcommands.
- capture.py – Capture adapters (FileAdapter, LiveAdapter) using PyShark, plus normalize_basics(pkt) and optional per-packet ops logging (log_packet_ops).
- logging.py – JSONL writers for detections and ops logs (adds schema_version to detections).
- signals.py – SIGINT handler returning a shared stop flag (STOP["flag"]).
- utils.py – Helpers (ISO timestamps, JSON dumps with orjson fallback, logs directory handling).

About capture_logger.py (Part 1):
In Week-1, you had the standalone capture_logger.py. For Milestone 2, its functionality is absorbed into:

- agent/capture.py (capture mechanics + normalization),
- agent/cli.py (dispatch + signals),
- agent/logging.py (structured JSONL).

Package: idsips/detectors/

- dns.py – Required. Suspicious DNS name heuristic (long labels, too long, high entropy).
- icmp.py – Required. Per-source ICMP rate in rolling 1s window.
- arp.py – Required. IP→{MACs} windowed mapping; ≥2 MACs for same IP triggers.
- http.py – Optional bonus. Simple keyword match in host/URI.

Package: idsips/siem/

- ingest.py – Robust line-oriented JSONL reader for detections.
- alerts.py – Writes alerts to alerts.jsonl and prints summaries.
- mini_siem.py – CLI with --timeline, --top, --rule-stats; basic correlation rules.

scripts/run_tests.py

Runs the visible tests:

1. DNS from PCAP → DNS_SUSPICIOUS present; ops shows start & shutdown
2. (Optional) ARP from PCAP → ARP_MULTIMAC present
3. Live --dry-run → SIGINT returns 0; ops shows shutdown
4. Mini-SIEM --rule-stats → returns 0; alerts file exists (may be empty)

Learning Resources:

These are some resources you can look up on the web and that can be used to understand concepts or code snippets that might be new to you. You're free to use your own resources as well, just don't use AI to write your code for you please.

- Python signal module - graceful shutdown patterns (official docs).
- JSON Lines format - jsonlines.org.
- DNS length limits - RFC 1035 (label ≤ 63 chars; full name ≤ 253 chars).
- ARP spoofing - common indicators and L2 background (standard networking texts).
- ICMP echo floods - rate limiting & DoS basics (standard networking texts).
- PyShark/Scapy (official documentation).