Bio-Adaptive IoT Honeynet — Team Documents (Router + CCTV)

Based on: Original Project Concept

Overview

This package contains short, shareable documents your team can use immediately. Each section below is self-contained and written to be copy-pasted into slides, readme files, or project reports.

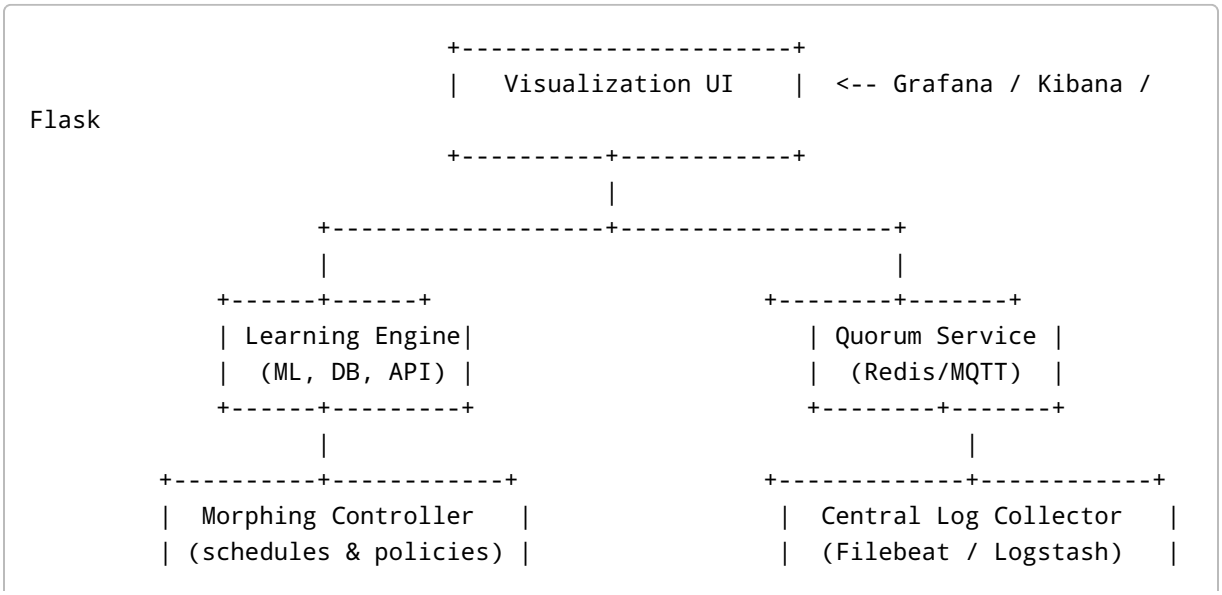
Sections: - A) Architecture diagram (text + ASCII and component list) - B) Dockerized setup (docker-compose + folder layout + environment notes) - C) Morphing engine design (logic, schedules, sample pseudocode) - D) Quorum system (message flow, thresholds, fault-tolerance) - E) Learning engine (ML pipeline, features, sample training snippet) - F) Dashboard design (data sources, suggested panels, alerting) - G) Full updated project concept (router + CCTV focused)

A) Architecture diagram

1-line summary

A distributed honeynet cluster that emulates a home router and an IP camera per node, connected to a central coordination plane (quorum, morphing controller, learning engine, and visualization).

ASCII diagram (shareable in plain text)



```
         +----------+------------+                 +-------------+------------+
                    |                                            |
    +---------------+--------------+  Network Overlay (VXLAN/VPN)
+-----------------+
    |                              |
<---------------------------------------------->   |
+---+---+                      +---+---
+                                        +---+---+
| Node1 |                      | NodeN
|                                        | Cloud |
|Router |                      |Router
|                                        |Storage|
|Camera |                      |Camera
|                                        +-------+
+---+---+                      +---+---+
    |                              |
    +---> Docker containers: cowrie, fake-http, rtsp-server, morph-agent,
metrics
```

Components (brief)

- **Nodes:** Lightweight VMs or Raspberry Pis running Docker. Each node hosts two device emulations: Router and Camera.
- **Morphing Controller:** Runs rules for periodic fingerprint changes and pushes device profiles to nodes.
- **Quorum Service:** Redis or MQTT topic bus used for cross-node alert validation.
- **Learning Engine:** Central ML pipeline — consumes parsed logs, trains models, issues signatures.
- **Log Collector:** Filebeat → Logstash → Elasticsearch, or a simple Fluentd pipeline.
- **Visualization:** Grafana/Kibana + custom Flask frontend for infection maps.

B) Dockerized setup

Goals

- Make it easy to deploy nodes with Docker Compose.
- Keep device emulators isolated, with persistent logging volumes.

Folder layout (repo root)

```
honeynet/
├─ nodes/
│  ├─ node-template/
│  │  ├─ docker-compose.yml
│  │  ├─ services/
```

```
│  │  │  ├─ cowrie/ (cowrie config)
│  │  │  ├─ fake-router-web/ (Flask/Node static UI)
│  │  │  ├─ rtsp-server/ (static jpeg / ffmpeg wrapper)
│  │  │  ├─ morph-agent/ (python)
│  │  │  └─ metrics-exporter/
├─ control/
│  ├─ morphing-controller/
│  ├─ quorum-service/
│  ├─ learning-engine/
│  └─ viz/
└─ docs/
```

Example `docker-compose.yml` (node-template)

```yaml
version: '3.7'
services:
  cowrie:
    image: cowrie/cowrie:latest
    container_name: cowrie
    restart: unless-stopped
    volumes:
      - ./services/cowrie/etc:/cowrie/etc
      - ./logs/cowrie:/cowrie/log
    networks:
      - honeynet
    ports:
      - "2222:2222"   # SSH/Telnet emulation (external mapped)
      - "2323:2323"

  fake-router-web:
    build: ./services/fake-router-web
    container_name: fake-router
    restart: unless-stopped
    volumes:
      - ./services/fake-router-web/static:/app/static
      - ./logs/fakerouter:/app/logs
    networks:
      - honeynet
    ports:
      - "8080:80"

  rtsp-server:
    build: ./services/rtsp-server
    container_name: rtsp
    restart: unless-stopped
    networks:
```

```
      - honeynet
    ports:
      - "1554:554"   # map internal RTSP 554 -> host 1554 to avoid collisions

  morph-agent:
    build: ./services/morph-agent
    container_name: morph-agent
    restart: on-failure
    environment:
      - MORPH_CONTROLLER_URL=http://control:5000
      - NODE_ID=node-01
    volumes:
      - ./config/morph:/etc/morph
      - ./logs/morph-agent:/var/log/morph
    networks:
      - honeynet

networks:
  honeynet:
    driver: bridge
```

Notes

- **Port mapping:** Map internal device ports to unique host ports so many nodes can run on a single host (use iptables/NAT or a reverse proxy for public exposure).
- **Persistence:** Mount logs to host volumes for central collection.
- **Security:** Run nodes in an isolated VPC, never expose management APIs to the public Internet.

C) Morphing engine design

Objective

Introduce controlled, rule-based variability in honeypot fingerprints so attackers cannot easily fingerprint and blacklist the honeypots.

Morphing vectors

- **Network identity:** change MAC OUI, IP TTL signature
- **Service banners:** ssh/telnet banners, HTTP server headers
- **HTTP UI:** different HTML templates, change resource paths
- **Open ports:** occasionally open/close optional ports (e.g., telnet on/off)
- **Filesystem:** change fake directory layout, config file contents
- **Timing:** change response latencies randomly within limits

Control model

- **Profiles:** `TP-Link-Archer-v1`, `Huawei-HG8245`, `Dahua-IPC1`, `Hikvision-DS2` — each profile contains a set of attribute overrides.
- **Policy engine:** chooses a profile per device according to a schedule or stochastic process.
- **Randomization:** small random perturbation applied to numeric attributes (TTL, delay) to avoid identical sessions.

Schedule & quorum-aware changes

- **Base frequency:** deterministic daily window (with jitter of +/- 0–3 hours).
- **Event-driven morphing:** if learning engine marks an IP as 'targeting honeypots', increase morphing frequency for 24h.
- **Coordinated morphing:** quorum may decide to rotate a subset of nodes simultaneously to observe propagation changes.

Sample pseudocode

```
# morph-agent (runs on node)
while True:
    profile = fetch_profile(controller_url, node_id)
    apply_profile(profile)
    wait(random_jitter(base_interval))

# controller side selects profile
if threat_level_high:
    select profiles with higher randomness
else:
    select normal rotation
```

Safety & ethics

- Keep morphing subtle enough to remain realistic.
- Do not morph to impersonate critical infrastructure or third-party brands in ways that cause harm.

D) Quorum system

Purpose

Cross-validate alerts from multiple nodes to reduce false positives and raise confidence before triggering high-impact responses.

Core components

- **Message bus:** Redis Pub/Sub or MQTT broker (e.g., Mosquitto)

- **Event format:** JSON with fields: `{ node_id, timestamp, event_type, confidence, features }`
- **Quorum engine:** subscribes to event topics and applies aggregation rules.

Example rules

1. **Temporal quorum:** at least `k` distinct nodes report `telnet_bruteforce` within `t` seconds -> raise cluster alert.
2. **Diverse-feature quorum:** when events include different protocols (e.g., RTSP + SSH) from same source IP -> high confidence.
3. **Weighted quorum:** each node has a trust weight; aggregated confidence = sum(node_weight * event_confidence).

Thresholds (starter values)

- `k = 2` nodes (for small clusters)
- `t = 60` seconds
- alert if aggregated_confidence >= 0.8

Fault tolerance

- Use persistent queues (Redis streams or MQTT with QoS) to avoid losing events.
- Implement leader election (e.g., via Redis locks) if quorum engine is replicated.

Response actions

- Create a ticket / log to the learning engine
- Trigger more aggressive morphing for the targeted nodes
- Flag source IP for sinkholing in DNS or firewall blocklists (manual review first)

E) Learning engine (ML)

Goal

Turn collected logs into evolving detection rules and anomaly scores.

Data pipeline

1. **Ingest:** Logstash/Fluentd or custom collector parses Cowrie, RTSP server, HTTP logs.
2. **Normalize:** Convert to a canonical event schema (timestamp, src_ip, dst_port, username, ua, uri, payload_hash, etc.).
3. **Feature store:** Produce time-windowed features per source IP (counts, unique usernames, avg inter-arrival, entropy of URIs).
4. **Training:** Use labeled historical data (if available) + unsupervised learning for anomalies.
5. **Deployment:** Export signatures/rules to the morphing controller and quorum service.

Suggested features

- `attempt_count_5m`, `unique_usernames_5m`, `avg_payload_size`, `rtsp_auth_fail_rate`, `ssh_banner_diff_score`, `unique_ports_scanned`

Models

- **Unsupervised:** Isolation Forest or DBSCAN for anomaly detection.
- **Supervised:** Random Forest / XGBoost for classification if labeled samples exist (e.g., Mirai vs benign scans).
- **Sequence models:** LSTM/GRU on login attempt sequences for more advanced detection.

Sample training snippet (scikit-learn)

```
from sklearn.ensemble import IsolationForest
clf = IsolationForest(n_estimators=100, contamination=0.01)
clf.fit(feature_matrix)
anomaly_scores = clf.decision_function(new_features)
```

Feedback loop

- New high-confidence detections are converted into signatures (IP blocklists, username lists, payload hashes) and distributed to nodes.
- Human-in-the-loop: require review for signature promotion to automatic enforcement.

F) Dashboard design

Data sources

- Elasticsearch (or InfluxDB + Prometheus) containing parsed events
- Redis for real-time quorum events
- Postgres (optional) for ML metadata and labeled events

Suggested panels (Grafana / Kibana)

- **Live map:** geo-IP of attack sources with time slider
- **Device timeline:** events per device (router vs camera) over time
- **Top attacker IPs:** with details and confidence scores
- **Protocol breakdown:** NTEL/Telnet/RTSP/HTTP counts
- **Quorum alerts:** list + expand details (nodes involved, timestamps)
- **Morphing history:** timeline showing profile changes per node
- **ML metrics:** anomaly rates, model drift indicators

Alerts

- Slack / Email on high-confidence quorum alerts
- Pager duty integration for critical incidents

Infection map concept

- Represent honeypots as cells; link attacker IPs to affected cells and show propagation (edges weighted by timestamp differences).

G) Full updated project concept (router + CCTV version)

Project Title: Bio-Adaptive Honeynet for IoT Threat Intelligence (Router + CCTV focus)

Short description: A distributed, adaptive honeynet cluster that emulates home routers and IP cameras. The system morphs device fingerprints over time, cross-validates suspicious behavior via quorum sensing, and evolves detection signatures via machine learning. Visualizations show infection propagation and the system's adaptive response.

Key objectives (recast): 1. Deploy an initial cluster of 3 nodes (each emulating a router + camera) using Docker. 2. Implement a morphing engine that rotates device fingerprints on a schedule and on-demand via quorum triggers. 3. Build a quorum-based alerting service to improve detection confidence. 4. Develop a learning engine that ingests logs, detects anomalies, and produces signatures. 5. Produce a visualization dashboard that maps attacker behavior and system adaptation.

Deliverables: - Dockerized node image + control plane - Morphing controller + policies - Quorum implementation (Redis/MQTT) and aggregator - ML pipeline with at least unsupervised anomaly detection - Visualization dashboard and final report (with dataset)

Ethics & safety: - Nodes run in an isolated network and use strict egress rules. - No real user traffic is intercepted; DNS redirect or sinkholing must be used only for research on consented traffic. - Human review required before blocking or sinkholing external IPs.

Timeline (12 weeks — adapted) - Weeks 1–3: Repo scaffold, Docker node templates, basic emulators (cowrie + fake web + rtsp) - Weeks 4–7: Morphing engine and quorum service; node orchestration - Weeks 8–10: Learning engine, model training, feedback loop - Weeks 11–12: Dashboard, evaluation, final report

---

How to use these documents

- Share this canvas with team members (it's organized for copy/paste into slides and READMEs).
- Implementation-ready bits: docker-compose snippet, ML starter code, morph-agent pseudocode.

If you want, I can: - Export each section into separate Markdown files. - Generate a slide deck (PowerPoint) summarizing these documents. - Produce a printable one-page summary for the project supervisor.

Document created for the team — edit and redistribute as needed.