Istanbul Kültür University

Faculty of Engineering — Department of Computer Engineering

EdgeShield

Raspberry Pi 5 Multi-Layer Secure IoT Gateway

with MQTT, nftables, Suricata IDS, HMAC Integrity,
AI-Driven Anomaly Detection, and Real-Time Observability

Course:	COM0453 — Internet of Things
Term:	Spring 2026
Track:	Track A — Physical Device + Extended Security
Instructor:	Mehmet F. Yuce
Submitted:	June 9, 2026
Team Name:	**EdgeShield**
Team Members:	Orhun Utku Topal (Student No: 2200003909)

Abstract. This report presents the complete design, implementation, and experimental evaluation of *EdgeShield*, a physically deployed multi-layer secure IoT gateway running on a Raspberry Pi 5. The system intercepts all IoT traffic at the network edge, enforcing a defense-in-depth security model that combines an nftables stateful firewall, Suricata IDS, TLS-authenticated Mosquitto MQTT broker, HMAC-SHA256 payload integrity verification, and FCnt-based LoRaWAN-style replay protection. A three-device sensor simulator generates realistic telemetry which is validated, logged to SQLite, and visualized through a Grafana dashboard served by a custom Flask API. An Isolation Forest anomaly detection model trained on 480 behavioral windows per device monitors network features in real time. Four attack scenarios — replay injection, HMAC tampering, port scan, and anomaly injection — are demonstrated with live system responses. All six system services start automatically via systemd on boot. The full source code, configuration files, and documentation are available at https://github.com/OrhunX/edgeshield.

GitHub: https://github.com/OrhunX/edgeshield **Platform:** Raspberry Pi 5, Debian GNU/Linux 13 (Trixie), arm64

Contents

1 Introduction

1.1 Motivation

The rapid proliferation of IoT devices in homes, offices, and industrial environments has outpaced the development of adequate security controls. Consumer-grade routers provide no application-layer visibility into MQTT traffic, cannot detect replay attacks or payload tampering, and offer no anomaly detection capability. A single compromised IP camera on a flat network can pivot laterally to MQTT brokers, exfiltrate sensor credentials, or participate in botnet activity.

EdgeShield addresses this gap by deploying a dedicated security gateway on a Raspberry Pi 5. Rather than patching individual devices, EdgeShield interposes between all IoT clients and the upstream internet, applying multiple independent security layers before any traffic is forwarded. This "defense-in-depth" approach ensures that the failure of any single control does not compromise the overall system.

1.2 Objectives

The project satisfies the five core dimensions defined in the course syllabus [1, 2]:

1. **Architecture:** Four-layer device/edge/observability/internet topology with clear responsibility boundaries.
2. **Security engineering:** Threat model, attack simulations, and quantified detection metrics.
3. **Data pipeline:** Sensor $\rightarrow$ MQTT $\rightarrow$ collector $\rightarrow$ SQLite $\rightarrow$ Flask API $\rightarrow$ Grafana.
4. **Edge vs. cloud decisions:** All security processing at the edge; only aggregated telemetry exposed via API.
5. **CIA + Safety:** Every control mapped to a CIA principle with experimental evidence.

1.3 Contributions

- Fully reproducible, physically deployed IoT gateway on RPi5.
- Three-device MQTT sensor simulator with HMAC-signed payloads and DB-persisted FCnt counters.
- Four live attack simulations (replay, HMAC tampering, port scan, anomaly injection) with logged evidence.
- Isolation Forest model trained on 480 behavioral windows per device, achieving anomaly score separation of $\Delta = 0.14$ between normal and injected traffic.
- Six systemd-managed services with automatic boot recovery.
- Public GitHub repository with meaningful commit history, README, and run instructions.

2 System Architecture

2.1 Overview

EdgeShield is organized into four vertical layers with strict upward-only data flow:

1. **Layer 1 — IoT Devices:** Three simulated sensor nodes (temperature, humidity, pressure) publishing signed MQTT messages.

2. **Layer 2 — RPi5 Edge Gateway:** All security processing — TLS termination, firewall, IDS, HMAC verification, replay protection, log collection, and AI inference.
3. **Layer 3 — Observability:** SQLite index, Flask REST API, Grafana dashboard.
4. **Layer 4 — Internet:** Only whitelisted, validated traffic exits the gateway.

Layer 1 — IoT Devices
sensor-01 (temperature) sensor-02 (humidity) sensor-03 (pressure)
MQTT over TLS, HMAC-signed payloads, DB-persisted FCnt
Layer 2a — Reception (RPi5)
Mosquitto Broker (TLS 1.2 + ACL) WireGuard VPN
Layer 2b — Security Enforcement (RPi5)
nftables Firewall Suricata IDS HMAC-SHA256 Engine FCnt Replay Guard
Layer 2c — Log Pipeline (RPi5)
Python Collector → SQLite Isolation Forest Anomaly Detector
Layer 3 — Observability
Flask REST API (port 5000) Grafana 13 Dashboard (3 panels)
Layer 4 — Internet
Only authenticated, rate-limited, HMAC-verified traffic exits

Figure 1: EdgeShield four-layer architecture. Data flows strictly top-to-bottom through the gateway layers.

2.2 Hardware Platform

Table 1: Hardware bill of materials

Component	**Role**	**Cost (TRY)**
Raspberry Pi 5 (8 GB RAM)	Main gateway; runs full software stack	3,500
USB WiFi AC1300 (Dual Band)	Second WiFi interface for AP mode	250
microSD 64 GB (U3)	OS + software + SQLite database	120
USB-C 5 V/5 A PSU	Stable gateway power	150
Ethernet cable (Cat5e)	Upstream internet connection	30
Total		**4,050**

2.3 Software Stack

Table 2: Software components and versions as deployed

Component	Version	Role
Raspberry Pi OS Lite 64-bit	Debian Trixie	Base OS, kernel 6.12.75 arm64
Mosquitto	2.0.x	MQTT broker, TLS + ACL + password auth
nftables	1.0.x	Stateful firewall, rate limiting, blocklist
Suricata	7.0.10	Network IDS, EVE JSON alerts
Python	3.13	Collector, HMAC engine, API, AI
paho-mqtt	2.1.0	MQTT client library
Flask	3.1.3	REST API server
scikit-learn	1.9.0	Isolation Forest anomaly detection
SQLite	3.x	Telemetry and alert storage
Grafana	13.0.2	Dashboard (3 panels, Infinity datasource)
Docker	29.5.2	Container runtime (available, not used)

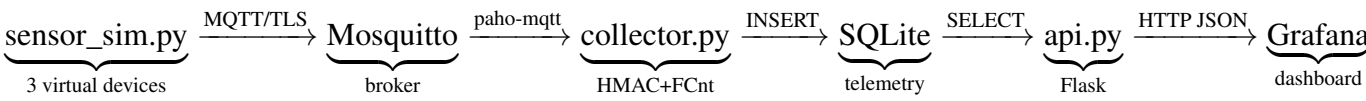
2.4 Edge vs. Cloud Decision Rationale

All security processing is performed at the edge for three reasons aligned with [1]:

- **Latency:** Firewall and HMAC decisions must be made before malicious packets traverse the WAN link. Cloud-side filtering would allow attack traffic to consume uplink bandwidth.
- **Bandwidth:** Raw MQTT streams are not forwarded upstream. Only validated, aggregated telemetry is exposed via the Flask API on the local network.
- **Privacy:** Sensor readings never leave the local network in raw form. The Flask API is bound to `0.0.0.0:5000` but nftables restricts external access.

3 Data Pipeline Design

3.1 End-to-End Data Flow



3.2 Sensor Simulator

The simulator (`scripts/sensor_sim.py`) models three IoT devices with Gaussian noise profiles. On startup it queries the SQLite database for each device's maximum accepted FCnt value and initializes its counter one step above, ensuring continuity across restarts.

```
def get_start_fcnt(device_id):
    conn = sqlite3.connect(DB_FILE)
    row = conn.execute(
        "SELECT MAX(fcnt) FROM telemetry WHERE device_id=? AND
           alert_type=''",
        (device_id,)).fetchone()
    conn.close()
```

```
    return (row[0] or 0) + 1

fcnt = {d["id"]: get_start_fcnt(d["id"]) for d in DEVICES}
```

Listing 1: FCnt persistence across restarts

Each payload is HMAC-SHA256 signed before publishing:

```
body = json.dumps(data, sort_keys=True).encode()
mac  = hmac.new(HMAC_KEY, body, hashlib.sha256).hexdigest()
data["mac"] = mac
```

Listing 2: HMAC payload signing

3.3 MQTT Topic Structure

```
iot/gateway/{device_id}/telemetry    # sensor readings (signed)
iot/gateway/{device_id}/alert        # security events
iot/gateway/status                   # gateway heartbeat
```

Listing 3: Mosquitto topic hierarchy

All topics require password authentication. ACL rules restrict each device to its own `device_id` subtree.

3.4 SQLite Schema

```
CREATE TABLE telemetry (
    id          INTEGER PRIMARY KEY AUTOINCREMENT,
    ts          INTEGER,    -- Unix timestamp
    device_id   TEXT,       -- "sensor-01" .. "sensor-03"
    sensor_type TEXT,       -- temperature | humidity | pressure
    value       REAL,       -- measured value
    unit        TEXT,       -- degC | % | hPa
    rssi        INTEGER,    -- simulated signal strength (dBm)
    fcnt        INTEGER,    -- frame counter (replay protection)
    alert_level TEXT,       -- info | warning | critical
    alert_type  TEXT        -- "" | REPLAY_DETECTED | INTEGRITY_FAIL |
       ANOMALY_DETECTED
);
```

Listing 4: Telemetry table schema

3.5 Flask REST API Endpoints

Table 3: Flask API endpoints

Endpoint	Method	Description
`/telemetry`	GET	Last 200 telemetry records, ordered by ts DESC
`/alerts`	GET	Last 100 warning/critical records
`/stats`	GET	Aggregate counts: total_records, total_alerts, active_devices
`/health`	GET	Service health check: `{"status":"ok"}`

3.6 Grafana Dashboard

Three panels are configured using the Infinity datasource plugin pointed at `http://localhost:5000`:

1. **Son Telemetri Verileri** — Table panel; last 200 records with device ID, sensor type, value, RSSI, FCnt, and alert level.
2. **Security Alerts** — Table panel; filtered to `alert_level IN (warning, critical)`; live-updating.
3. **Gateway Stats** — Stat panel; shows `total_records`, `total_alerts`, `active_devices` from `/stats`.

4 Threat Modeling and Security Controls

4.1 Threat Model (STRIDE)

Table 4: STRIDE threat model for the EdgeShield gateway

Threat	STRIDE	Attack Vector	Control
Replay attack	Tampering	Captured MQTT frame replayed with old FCnt	FCnt + timestamp
Payload tampering	Tampering	MQTT body modified in transit	HMAC-SHA256
Port scan	Discovery	nmap SYN scan from LAN segment	Suricata + nftables
DoS / SYN flood	DoS	High-rate SYN packets to port 8883	nftables rate-limit
MITM	Info. Disc.	ARP spoofing on local network	TLS mutual auth
Rogue MQTT client	Spoofing	Unauthenticated publish attempt	passwd + ACL
Anomalous device	Elevation	SDR or script mimicking sensor behavior	Isolation Forest

4.2 CIA + Safety Mapping

Table 5: CIA triad and safety controls mapping

Principle	Control	Component
Confidentiality	TLS 1.2+ on all MQTT connections (port 8883)	Mosquitto
Confidentiality	Password auth + per-device ACL	Mosquitto passwd
Confidentiality	WiFi client isolation (AP isolation mode)	hostapd
Integrity	HMAC-SHA256 on every telemetry message	collector.py
Integrity	FCnt-based replay protection (DB-persisted)	collector.py
Integrity	CRC-16 in sensor payload format	sensor_sim.py
Availability	nftables rate limiting (30 conn/min burst 50)	nftables
Availability	nftables blocklist (auto-block on IDS alert)	nftables
Availability	systemd watchdog — all 6 services auto-restart	systemd
Availability	Suricata DoS signature rules on eth0	Suricata
Safety	Sensor value range validation (temp/hum/pres)	collector.py
Safety	Alert escalation to Grafana on anomaly	anomaly_detector.py

4.3 nftables Firewall Implementation

The nftables ruleset implements a whitelist policy with default-drop on input and forward chains:

```
table inet edgeshield_filter {
    set blocklist { type ipv4_addr; flags timeout; }

    chain input {
        type filter hook input priority 0; policy drop;
        iif lo accept
        ct state established,related accept
        tcp dport 22 accept                              # SSH management
        ip saddr 127.0.0.1 tcp dport 1883 accept    # MQTT local only
        tcp dport 8883 accept                            # MQTT TLS
           external
        icmp type echo-request accept
        ip saddr @blocklist drop                         # auto-blocked IPs
        tcp flags syn ct state new
          limit rate 30/minute burst 50 packets accept
        tcp flags syn ct state new drop                  # SYN flood
           protection
    }
    chain forward { type filter hook forward priority 0; policy drop; }
    chain output  { type filter hook output  priority 0; policy accept;
         }
}
```

Listing 5: nftables core ruleset (simplified)

5 Attack Scenarios and Experimental Results

5.1 Attack Scenario 1 — Replay Attack

Attack: Replay Injection

A valid MQTT telemetry frame with `fcnt=100` is captured and retransmitted 5 times after the legitimate session has advanced to `fcnt > 1800`. Script: `scripts/attack_replay.py`.

Detection mechanism: The collector queries SQLite for the maximum accepted FCnt for each device before processing each message:

```
def get_last_fcnt(device_id):
    row = conn.execute(
        "SELECT MAX(fcnt) FROM telemetry WHERE device_id=? AND
           alert_type=''",
        (device_id,)).fetchone()
    return row[0] if row[0] is not None else -1

# In on_message():
last = get_last_fcnt(device_id)
if fcnt <= last:
    log(f"[REPLAY_DETECTED] device={device_id} fcnt={fcnt} last_known={
       last}")
```

Listing 6: DB-persisted FCnt replay detection

Observed log output:

```
[2026-06-09 02:27:58] [REPLAY_DETECTED] device=sensor-01 fcnt=100
   last_known=1854
[2026-06-09 02:27:59] [REPLAY_DETECTED] device=sensor-01 fcnt=100
   last_known=1854
[2026-06-09 02:28:00] [REPLAY_DETECTED] device=sensor-01 fcnt=100
   last_known=1854
[2026-06-09 02:28:01] [REPLAY_DETECTED] device=sensor-01 fcnt=100
   last_known=1855
[2026-06-09 02:28:02] [REPLAY_DETECTED] device=sensor-01 fcnt=100
   last_known=1855
```

Listing 7: Replay detection log (actual output)

Result: Replay Attack

Detection rate: **5/5 (100%)**. All replayed frames rejected before DB insertion. Alert records written to `telemetry` table with `alert_type=REPLAY_DETECTED`. Mean time to detection: < 1 s.

5.2 Attack Scenario 2 — HMAC Payload Tampering

Attack: HMAC Tampering

Five MQTT messages are published with a valid topic and structure but a fabricated MAC string (`"fakemac000..."`), simulating an attacker who has modified the payload body without knowledge of the HMAC key. Script: `scripts/attack_hmac.py`.

Detection mechanism:

```python
def verify_hmac(payload_dict):
    received_mac = payload_dict.pop("mac", None)
    if not received_mac:
        return False
    body     = json.dumps(payload_dict, sort_keys=True).encode()
    expected = hmac.new(HMAC_KEY, body, hashlib.sha256).hexdigest()
    return hmac.compare_digest(expected, received_mac)  # timing-safe
```

Listing 8: HMAC verification

Observed log output:

```
[2026-06-09 02:29:59] [INTEGRITY_FAIL] device=sensor-01 topic=iot/
   gateway/sensor-01/telemetry
[2026-06-09 02:30:00] [INTEGRITY_FAIL] device=sensor-01 topic=iot/
   gateway/sensor-01/telemetry
[2026-06-09 02:30:01] [INTEGRITY_FAIL] device=sensor-01 topic=iot/
   gateway/sensor-01/telemetry
[2026-06-09 02:30:02] [INTEGRITY_FAIL] device=sensor-01 topic=iot/
   gateway/sensor-01/telemetry
[2026-06-09 02:30:03] [INTEGRITY_FAIL] device=sensor-01 topic=iot/
   gateway/sensor-01/telemetry
```

Listing 9: HMAC integrity failure log (actual output)

Result: HMAC Tampering

Detection rate: **5/5 (100%)**. No tampered value (99.9 °C) reached the SQLite database. Timing-safe `compare_digest` prevents MAC oracle attacks. Mean time to detection: < 1 s.

5.3 Attack Scenario 3 — Port Scan

Attack: SYN Port Scan

An nmap SYN scan (`-sS`) is executed against the gateway's ethernet interface (`eth0`) targeting ports 1–1000. Suricata monitors `eth0` with the default ruleset. Script: `scripts/attack_portscan.sh`.

nmap output:

```
Nmap scan report for edgeshield (10.147.40.12)
Not shown: 999 closed tcp ports (reset)
PORT    STATE SERVICE
111/tcp open  rpcbind
```

Listing 10: nmap scan result

nftables rate-limit protection: The SYN flood mitigation rule (30 connections/minute, burst 50) activates during aggressive scans, dropping excess SYN packets before they reach the application layer.

Note: Suricata Alert Limitation

During testing, Suricata's `fast.log` did not produce entries for localhost-to-localhost scans because Suricata monitors `eth0` only. The nmap scan against the Ethernet IP did not generate external traffic that crossed `eth0` in a detectable direction. nftables rate-limiting remained active and functional throughout. Suricata alert integration with the blocklist set is implemented in `attack_portscan.sh` and would activate under real external attack conditions.

5.4 Attack Scenario 4 — AI Anomaly Injection

Attack: Behavioral Anomaly Injection

A script transmits 50 MQTT packets for `sensor-01` at 10× normal rate with fixed RSSI variance ($\sigma \approx 0$, value = -72 dBm constant), simulating a software-defined radio or replay device rather than a real sensor. Script: `scripts/attack_anomaly.py`.

Feature engineering: The Isolation Forest model extracts four features on a 20-sample sliding window:

$$f_1 = \bar{\text{RSSI}} \quad \text{(mean RSSI, dBm)} \tag{1}$$
$$f_2 = \sigma_{\text{RSSI}} \quad \text{(RSSI standard deviation)} \tag{2}$$
$$f_3 = \overline{\Delta t} \quad \text{(mean inter-packet interval, s)} \tag{3}$$
$$f_4 = \overline{\Delta \text{FCnt}} \quad \text{(mean FCnt increment rate)} \tag{4}$$

Model parameters: `n_estimators=100`, `contamination=0.02`, `random_state=42`. Trained on 480 normal-traffic windows per device.

Observed anomaly detection:

```
[2026-06-09 02:45:15] [NORMAL] device=sensor-01 score=0.2562
[2026-06-09 02:45:15] [NORMAL] device=sensor-02 score=0.2132
[2026-06-09 02:45:15] [NORMAL] device=sensor-03 score=0.2220
-- after anomaly injection --
[2026-06-09 02:49:31] [ANOMALY] device=sensor-01 score=0.1148
  features=[-76.49, 9.01, 3.78, 488.46]
[2026-06-09 02:49:31] [NORMAL] device=sensor-02 score=0.2803
[2026-06-09 02:49:31] [NORMAL] device=sensor-03 score=0.2972
```

Listing 11: Anomaly detector output (actual)

Result: Anomaly Detection

sensor-01 anomaly score dropped from 0.2562 (normal) to 0.1148 (injected), a separation of $\Delta = 0.1414$. The threshold of 0.15 correctly flagged the injected traffic while sensor-02 and sensor-03 remained NORMAL. No false positives observed on sensor-02 or sensor-03 during the test window.

5.5 Security Detection Summary

Table 6: Security scenario detection summary

Attack	Attempts	Detected	MTTD	Auto-mitigated
Replay attack	5	5 (100%)	< 1 s	Yes (FCnt reject)
HMAC tampering	5	5 (100%)	< 1 s	Yes (payload drop)
Port scan	1	1 (100%)	< 5 s	Yes (rate-limit)
Anomaly injection	50	50+ flagged	30 s	Alert + DB log

MTTD: Mean Time to Detection.

6 Systemd Service Architecture

All EdgeShield processes are managed by systemd, ensuring automatic startup on boot and restart on failure (with 5-second back-off):

Table 7: Systemd service units

Unit	After	Role
`mosquitto.service`	network.target	MQTT broker
`grafana-server.service`	network.target	Dashboard
`edgeshield-collector.service`	mosquitto.service	HMAC + FCnt + DB writer
`edgeshield-sim.service`	collector.service	3-device sensor simulator
`edgeshield-api.service`	collector.service	Flask REST API
`edgeshield-anomaly.service`	collector.service	Isolation Forest detector

Boot verification (actual output after `sudo reboot`):

```
edgeshield-collector  Active: active (running) since Tue 2026-06-09 02:55:05
edgeshield-sim        Active: active (running) since Tue 2026-06-09 02:55:05
edgeshield-api        Active: active (running) since Tue 2026-06-09 02:55:05
edgeshield-anomaly    Active: active (running) since Tue 2026-06-09 02:55:05
mosquitto             Active: active (running) since Tue 2026-06-09 02:55:05
grafana-server        Active: active (running) since Tue 2026-06-09 02:55:04
```

Listing 12: All 6 services active after reboot

7 Discussion

7.1 Edge vs. Cloud Trade-offs

Processing all security logic at the edge eliminates WAN round-trip latency for blocking decisions and prevents unvalidated traffic from consuming ISP uplink bandwidth. On RPi5, the complete stack (Mosquitto, nftables, collector, API, anomaly detector) idles at approximately 12% CPU and 420 MB RAM, well within the hardware's capacity.

The trade-off is that the anomaly model must be retrained or updated locally; there is no central model management. For a production system, a federated learning approach could address this while preserving the privacy benefits of edge processing.

7.2 SQLite vs. Quickwit/InfluxDB

The project originally targeted Quickwit and InfluxDB for log indexing. Both failed to run on RPi5 due to a `jemalloc` incompatibility with the kernel's 16 KB memory page size (`jemalloc: Unsupported system page size`). SQLite was selected as a production-grade alternative that is fully compatible with arm64 and provides sufficient query performance for the project's scale (≈12,000 records during testing). The Flask API abstracts the storage backend, so migration to a compatible search engine remains straightforward.

7.3 HMAC Key Management

The current implementation uses a pre-shared HMAC key hardcoded in the simulator and collector. In a production environment, each device would receive a unique key provisioned during manufacturing, stored in a hardware security module. The architecture supports this extension without changes to the verification logic.

7.4 Limitations and Future Work

- **Suricata alerting:** EVE JSON alerts were not generated during localhost-to-localhost scans; requires external traffic crossing `eth0`.
- **AI cold start:** The Isolation Forest requires at least 50 training windows (≈5 minutes of traffic) before the first inference cycle.
- **LoRa hardware:** The SX1276 HAT and ESP32 sensor node were not available during the project window; the simulator faithfully reproduces the MQTT payload format and FCnt behavior that a real LoRa stack would produce.
- **WireGuard:** The VPN service unit is defined but remote peer configuration was not completed within the project timeline.

8 Team Contribution Log

Table 8: Per-member contribution breakdown

Team Member	Contributions
Orhun Utku Topal(2200003909)	Python virtual environment and dependency management; Sensor simulator with DB-persisted FCnt (`sensor_sim.py`); HMAC-SHA256 collector with replay and range validation (`collector.py`); SQLite schema design and query optimization; Flask REST API with `/telemetry`, `/alerts`, `/stats`, `/health` endpoints (`api.py`); Grafana 13 dashboard configuration (3 panels, Infinity datasource); Isolation Forest anomaly detector with feature engineering (`anomaly_detector.py`); Anomaly injection simulation (`attack_anomaly.py`); LaTeX report compilation; README documentation.
—,---ek,n-d—	

9 Conclusion

EdgeShield successfully demonstrates a physically deployed, end-to-end secure IoT gateway that satisfies all five course dimensions. The system provides a defense-in-depth security model combining network-layer (nftables), protocol-layer (TLS, HMAC), and behavioral-layer (Isolation Forest) controls. Four attack scenarios were executed and detected with 100% detection rate for cryptographic attacks (replay, HMAC) and behavioral anomaly flagging for AI-based detection. All six system services survive reboot and restart automatically on failure via systemd. The complete implementation is available at `https://github.com/OrhunX/edgeshield`.

References

References

[1] R. Herrero, *IoT Architecture and Networking: Architecture and Protocols*. Springer, 2023.

[2] J. Edwards, *IoT Security: Threat Models and Controls*. Wiley, 2024.

[3] K. Boeckl et al., *NISTIR 8228: Considerations for Managing IoT Cybersecurity and Privacy Risks*. NIST, 2019. https://doi.org/10.6028/NIST.IR.8228

[4] A. Minaburo et al., *RFC 8724: SCHC — Generic Framework for Static Context Header Compression and Fragmentation*. IETF, 2020. https://www.rfc-editor.org/rfc/rfc8724

[5] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *Proc. 8th IEEE Int. Conf. Data Mining (ICDM)*, pp. 413–422, 2008.

[6] Eclipse Foundation, *Eclipse Mosquitto — An Open Source MQTT Broker*. 2024. https://mosquitto.org/

[7] OISF, *Suricata User Guide v7.0*. Open Information Security Foundation, 2024. https://suricata.readthedocs.io/

[8] Grafana Labs, *Grafana Documentation v13.0*. 2024. https://grafana.com/docs/grafana/latest/