



AI-Enhanced Security Assessment and Forensic Analysis of GraphQL APIs in Drone Fleet Management Systems

by

Muhammad Waleed

Student ID: 2430361

Supervisor: Dr Dipo Dunsin

Project report submitted as part of the requirements for the award of

MSc Cybersecurity and Digital Forensics

University of Wales Trinity Saint David

2025/26

Declaration and Consent

Declaration

I, Muhammad Waleed, declare that I am the author of this project; that all references cited have been consulted; that I have conducted the work described in this report; and that the finished work lies within the prescribed word limit. This work has not previously been accepted as part of another degree submission.

Signature: _____ Date: _____

Form of Consent

I, Muhammad Waleed, consent to my project, submitted for the award of MSc Cybersecurity and Digital Forensics, being made available for inter-library loan or photocopying if successful, subject to copyright law and university policy. I also consent to the title and abstract being made available to outside organisations.

Signature: _____ Date: _____

Abstract

Drone fleet platforms combine operational data, mission control, telemetry, users, and security-sensitive actions through software interfaces. GraphQL is attractive in this setting because a client can request connected records through a single typed query. The same flexibility also creates a security problem: deeply nested queries, high-cost operations, introspection, batching, excessive data access, and weak resolver authorisation may expose operational information or consume server resources. Existing research usually examines drone operations, GraphQL API security, machine-learning detection, and digital forensics as separate subjects. This project investigates whether these areas can be combined in one reproducible prototype.

The study used a design-science and experimental approach to create a local drone fleet management system with FastAPI, Strawberry GraphQL, SQLAlchemy, SQLite, and a browser-based dashboard. Authentication and access control were implemented through expiring HMAC-signed tokens, four role levels, password hashing, and resolver-level role checks. Protective controls included query-depth and complexity limits, login and API rate limits, and controlled attack simulation. Evidence support included automatic security logging, a SHA-256 evidence hash chain, forensic timeline reconstruction, and CSV/PDF reporting. A 300-record mixed dataset was used to compare an explainable rule-based detector with Decision Tree, Random Forest, Logistic Regression, Extra Trees, and Isolation Forest models. Leakage-prone outcome variables were excluded from model inputs.

Twenty-four automated tests passed. Live checks confirmed protected GraphQL access, role restrictions, introspection control, evidence export, report generation, and integrity verification. The original evaluation snapshot verified all 617 evidence records; a later isolated follow-up capture verified 632 out of 632. Supervised models achieved perfect hold-out scores on the synthetic dataset, while Isolation Forest achieved 0.4779 F1 and produced many false positives. The perfect supervised results are treated cautiously because the dataset is small, generated, and strongly separable. The main contribution is not a claim of production-ready detection accuracy, but an integrated and inspectable framework connecting API prevention, behavioural detection, and forensic readiness. Future work should evaluate real traffic, larger fleets, independent datasets, stronger key management, and externally validated attack traces.

Acknowledgements

I would like to thank my supervisor, Dr Dipo Dunsin, for guidance throughout the project. I also acknowledge the support of the MSc Cybersecurity and Digital Forensics teaching team and the authors and organisations whose research, standards, and security guidance informed this work.

Table of Contents

Declaration and Consent	ii
Declaration	ii
Form of Consent.....	ii
Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Figures	viii
List of Tables.....	ix
Chapter 1: Introduction	1
1.1 Background	1
1.2 Problem Statement	2
1.3 Aim and Objectives.....	2
1.4 Research Questions	3
1.5 Scope and Boundaries	3
1.6 Research Contribution.....	4
1.7 Dissertation Structure.....	4
Chapter 2: Literature Review	5
2.1 Introduction	5
2.2 Drone Fleet Management as a Software and Operational Problem	6
2.3 Reliability, Availability and Trust.....	8
2.4 GraphQL APIs and the Shift from Endpoints to Fields	9
2.5 GraphQL Security Risks in a Drone Fleet Context.....	10
2.6 AI-Based Anomaly Detection for API Behaviour	12
2.7 Digital Forensic Logging and Incident Reconstruction	14
2.8 Combined Discussion and Research Gap	15
2.9 Critical Comparison of the Main Literature.....	17

2.10 Limitations in Existing Research	19
2.11 Chapter Summary.....	21
Chapter 3: Research Methodology	22
3.1 Research Design.....	22
3.2 Artefact and Experimental Environment	22
3.3 Dataset Construction	23
3.4 Security Assessment Scenarios	23
3.5 Detection Methods	24
3.6 Evaluation Metrics	24
3.7 Reproducibility, Ethics, and Validity.....	25
3.8 Methodological Rationale and Alternatives.....	26
Chapter 4: System Requirements and Design.....	27
4.1 Requirements.....	27
4.2 Architecture.....	27
4.3 Authentication and Authorisation Design.....	28
4.4 GraphQL Abuse-Control Design	29
4.5 Evidence and Forensic Design.....	29
4.6 Attack and Detection Design	30
4.7 User Interface Design.....	30
4.8 Trust Boundaries and Failure Handling	31
Chapter 5: Implementation.....	32
5.1 Technology Stack.....	32
5.2 Operational GraphQL Schema	32
5.3 Request Security Implementation	33
5.4 Audit and Evidence Implementation.....	34
5.5 Rule-Based and ML Implementation.....	35
5.6 Forensics and Reporting Implementation	36
5.7 Deployment and Reproducibility	36

Chapter 6: Evaluation and Discussion	37
6.1 Evaluation Procedure	37
6.2 Functional and Security Results.....	38
6.3 Rule-Based Detection Results.....	39
6.4 Machine-Learning Results	39
6.5 Unseen Operation-Family Evaluation.....	41
6.6 Forensic Evaluation.....	42
6.7 Difference from Previous Research	43
6.8 Limitations	44
6.9 Integrated Discussion by Research Question	44
Chapter 7: Conclusions and Recommendations.....	46
7.1 Conclusions	46
7.2 Recommendations	46
7.3 Future Publication Direction	47
Chapter 8: Reflection	48
References	50
Appendix A: Principal GraphQL Security Tests	52
Appendix B: Default Security Configuration	53
Appendix C: Reproduction Commands	54
Appendix D: Word Count Statement	55
Appendix E: Real-Time Interface and Testing Screenshots	56

List of Figures

Figure 4.1 Implemented GraphQL security and evidence architecture

Figure 5.1 Live local API verification from the isolated evidence copy

Figure 5.2 Observed GraphQL security enforcement responses

Figure 5.3 Forensic reconstruction and evidence-chain verification

Figure 6.1 Fresh automated regression-test run

Figure 6.2 Detection precision, recall, and F1 comparison

Figure 6.3 Confusion matrices for three contrasting detectors

Figure 6.4 Leave-one-operation-family-out Random Forest evaluation

Figure A.1 Fresh isolated validation summary from actual pytest and API responses

Figure E.1 Authenticated Aegis Flight security overview with live fleet and security counters

Figure E.2 Controlled GraphQL Attack Laboratory after execution of all six simulation scenarios

Figure E.3 Automated complete-assessment workflow used for the supervisor demonstration

Figure E.4 AI/ML detection workspace after Extra Trees classification of accumulated local logs

Figure E.5 Forensic reconstruction with verified evidence chain and chronological incident records

Figure E.6 Evidence-log interface filtered to anomalous GraphQL activity

List of Tables

Table 1.1 Research objectives and questions

Table 3.1 Dataset composition

Table 3.2 Evaluation metrics

Table 4.1 Functional and security requirements

Table 4.2 Controlled attack scenarios

Table 5.1 Technology stack

Table 6.1 Verification summary

Table 6.2 Machine-learning evaluation results

Table 6.3 Leave-one-operation-family-out results

Table 6.4 Comparison between previous research and this project

Table B.1 Default local configuration

Chapter 1: Introduction

1.1 Background

Unmanned aerial vehicles are increasingly managed as connected software assets rather than isolated aircraft. A fleet platform may store airframe details, battery condition, location, assigned missions, telemetry, user roles, operational alerts, and historical records. Operators expect this information to be available through dashboards and application programming interfaces. Consequently, the security of the API layer can influence the confidentiality of locations and routes, the integrity of mission assignments, and the availability of services used to monitor a fleet. The project does not control real aircraft, but it models this software boundary because it is a realistic point at which operational and cyber risks meet.

GraphQL provides a typed query language and an execution model in which clients specify the fields they require. This is useful for connected drone data: a dashboard can request drones, missions, telemetry, and related user information without calling many fixed endpoints. However, field-level flexibility transfers substantial control to the client. A valid request can still be abusive if it is too deep, too complex, repeated too frequently, or asks for data outside the caller's role. For that reason, security cannot rely only on network filtering or a successful login. It must examine the structure and purpose of each operation before and during execution.

The second concern is visibility after an event. A conventional application log may record an HTTP request but omit the selected GraphQL fields, calculated depth, operation name, user role, affected asset, authorisation result, or evidence-integrity information. These omissions make later reconstruction difficult. A security control may block a request successfully while leaving insufficient evidence to explain what happened. Forensic readiness requires logs designed for investigation from the start, with consistent timestamps, contextual fields, integrity checks, and a way to reconstruct the sequence of actions.

The third concern is the responsible use of anomaly detection. Rule-based detection is understandable but can miss patterns that do not meet predefined thresholds. Machine learning can identify combinations of behaviour across query depth, complexity, response time, role, target identifiers, and operation type. Yet apparently strong accuracy can be misleading when labels leak into features, when the test split resembles

the training split, or when synthetic attacks have simple signatures. The research evaluates both transparent rules and multiple machine-learning models while treating generalisation as a separate question from hold-out accuracy.

1.2 Problem Statement

The central problem is fragmentation. Drone fleet studies commonly focus on routing, scheduling, reliability, batteries, or mission allocation. GraphQL studies examine adoption, performance, testing, and API-specific abuse. Cybersecurity research develops anomaly detection, while digital-forensic research studies log analysis and evidence handling. Each area contributes part of the solution, but a fleet operator still needs an integrated process: prevent unsafe API activity, retain meaningful evidence, detect suspicious behaviour, and reconstruct an incident using the same data trail.

A further practical problem is that security demonstrations often show attacks without showing how controls are enforced, or they report a model score without explaining the features, dataset limitations, and effect on evidence. Conversely, forensic demonstrations may preserve logs that were not captured at the API layer where the important decisions occurred. The project addresses this gap through one local prototype in which operational GraphQL requests, rejected requests, simulated attacks, anomaly predictions, and forensic reports share a common evidence model.

1.3 Aim and Objectives

The aim of this project is to design, implement, and evaluate an AI-enhanced security assessment and forensic-analysis framework for GraphQL APIs in a simulated drone fleet management system.

- Review research on drone fleet software, GraphQL APIs, API security, anomaly detection, and forensic log analysis to identify a defensible integration gap.
- Build a functional local prototype for users, drones, missions, telemetry, GraphQL operations, security monitoring, and evidence reporting.
- Implement and test authentication, role-based access control, query-depth and complexity controls, rate limits, introspection restrictions, and tamper-evident logging.
- Compare explainable rules with supervised and unsupervised machine-learning models using leakage-controlled behavioural features.

- Evaluate the prototype through automated tests, live role/security checks, model metrics, unseen-operation-family testing, and evidence-chain verification.

1.4 Research Questions

Table 1.1 Research objectives and questions

Research question	Purpose
RQ1: Which GraphQL-specific risks are most relevant to a drone fleet management API?	Connect API behaviour to fleet confidentiality, integrity, and availability.
RQ2: How can preventive controls and forensic-ready logging be integrated in one GraphQL request path?	Evaluate a design that records both accepted and rejected operations.
RQ3: How effectively do rules and selected ML models distinguish normal from anomalous API behaviour in the project dataset?	Compare precision, recall, F1, error rates, and generalisation checks.
RQ4: What evidence can the prototype provide for incident reconstruction and integrity verification?	Assess timelines, actor/asset context, exports, and the SHA-256 chain.

1.5 Scope and Boundaries

The artefact is a safe local proof of concept. It does not connect to physical drones, radio links, aviation-control systems, commercial cloud services, or live organisational accounts. Attack scenarios create controlled records or send requests only to the local API. The system models four roles - administrator, operator, analyst, and viewer - and supports fleet, mission, telemetry, security, detection, and reporting workflows. SQLite is used for reproducibility rather than production scale.

The study evaluates application-layer GraphQL controls and evidence. It does not claim to test wireless protocol security, GPS spoofing, physical capture, firmware compromise, or aircraft safety certification. It also does not claim that a 300-record synthetic dataset represents global attack behaviour. These boundaries are deliberate: they keep the experiment ethical, repeatable, and closely aligned with the research question.

1.6 Research Contribution

The contribution is an integrated, inspectable prototype rather than a new cryptographic primitive or learning algorithm. Its novelty at MSc level lies in connecting controls that are often demonstrated separately. The same GraphQL operation is analysed for depth and cost, checked against authentication and role rules, recorded as structured evidence, enriched by detection, and available to a forensic timeline. Outcome-derived fields are deliberately excluded from ML inputs, and simulated evidence is distinguishable from observed activity. These decisions make the evaluation easier to audit and expose weaknesses that a headline accuracy score might conceal.

The project also contributes a reproducible assessment package: pinned dependencies, Docker configuration, test cases, generated model artifacts, prediction files, explainability output, unseen-operation-family results, CSV evidence export, and a PDF assessment report. This package supports supervision, demonstration, and future replication without requiring a real drone environment.

1.7 Dissertation Structure

Chapter 2 critically reviews the research that defines the operational, API-security, machine-learning, and forensic context. Chapter 3 explains the design-science and experimental methodology, dataset, metrics, ethics, and validity controls. Chapter 4 presents requirements and system design. Chapter 5 explains implementation decisions. Chapter 6 evaluates functional security, detection performance, evidence integrity, and the difference between this artefact and earlier research. Chapter 7 concludes and recommends future work. Chapter 8 reflects on the research and development process.

Chapter 2: Literature Review

2.1 Introduction

This review is written for the topic of AI-enhanced security assessment and forensic analysis of GraphQL APIs in drone fleet management systems. The initial reading suggested a focus mainly on drones and AI. Closer examination of the literature showed that the API layer is also an important part of the problem. Drone fleet management, GraphQL security, machine-learning detection and digital forensics are not separate in this topic. They meet at the point where users send requests, missions are changed, telemetry is viewed and logs are created.

This also explains why the literature review is not arranged as a simple paper-by-paper summary. A paper-by-paper summary would make the argument flat and repetitive. The sources are considered through one main question: what do they reveal about the data, access rights, monitoring needs and investigation problems of a drone fleet platform? Some papers answer this directly, while others help indirectly by showing how missions, routes, batteries or reliability are managed.

This creates a clear boundary in the review. The drone literature explains the environment, the GraphQL literature explains the access problem, and the forensic literature explains what has to remain after the incident.

The general problem is that drone fleets are becoming more software-driven. Drones are no longer just remote-control flying machines. In many applications they are attached to mission planners, scheduling tools, routing algorithms, reliability models, dashboards, data storage, and sometimes AI-based decision modules. In such a platform, the API is the place where commands and data are exchanged. If the API is weak, then the weakness can affect more than a website page. It can expose drone telemetry, mission planning details, infrastructure inspection points, delivery routes, user roles, or even operational logs.

GraphQL fits this setting because drone fleet data is naturally connected. One drone is linked with missions, telemetry, batteries, alerts, operators, payloads and logs. A dashboard usually needs these records in different combinations depending on the user role. GraphQL gives a flexible way to request nested data, but the same flexibility creates security problems that are different from normal REST endpoint testing. Query

depth, field-level access, resolver checks, introspection and query complexity all become part of the security work.

The forensic side also needs attention. Many systems collect logs, but those logs are often created for debugging rather than investigation. After an incident, an analyst will ask practical questions: who made the request, what operation was run, which drone or mission was touched, and whether the action was authorised. The log also has to show enough context to judge whether the request was abnormal and whether the record itself can be trusted. For that reason, this review treats GraphQL testing, AI anomaly detection and forensic logging as connected parts of one problem.

The review remains close to what can be implemented within this project. Some papers are very advanced and use optimisation, digital twins or full fleet management platforms. Those works are useful, but the practical implementation has a smaller scope. The useful lesson is that a drone fleet system is full of linked data and role-based actions. That is enough to create a serious API security and forensic problem without bringing real drones into the lab.

2.2 Drone Fleet Management as a Software and Operational Problem

A useful starting point is that drone fleet management is not only about flying drones. It is about coordinating many decisions at the same time. In disaster relief, for example, drones deliver lightweight relief items to locations that cannot be reached easily by road, and Rabta et al. model this problem by considering payload, energy constraints and recharging stations [1]. From a software point of view, these mission details become data. Package priority, route, battery condition and destination are all records that a fleet platform has to process and protect.

Transportation and logistics reviews give a wider view of the same issue. Turgut and Şeker review drone technology in transportation management and discuss faster delivery, accessibility, efficiency and customer satisfaction, while also showing that drone logistics is still not mature and has technical and operational gaps [2]. Jazairy et al. review last-mile delivery from a logistics management perspective and consider different stakeholders, including senders, receivers, regulators and society [3]. In API terms, these stakeholders do not need the same level of access. A regulator, an operator, a customer and a maintenance user should see different parts of the system.

Industrial fleet papers show that the problem is not only outdoor delivery. Hažík et al. design a fleet management system for industrial mobile robots using ROS, hybrid maps,

and synchronisation logic [4]. Even though their work is on AMR robots and not flying drones, the fleet-management ideas are similar: there is a supervisor, there are routes, there is a shared environment, and there is a need to avoid conflict between moving agents. Olivares and Cordova study drone fleet management inside production systems, where drones perform pickup and delivery of materials and where battery replacement can interrupt production [5]. Later, Córdova and Olivares extend this line of work for a customised and flexible production system, where route generation and energy efficiency are main concerns [6].

These industrial studies are useful because they make the data model more realistic. A fleet platform stores status records, task requests, route assignments, battery levels, task priorities and sometimes plant layout or warehouse information. Exposure of this data through a weak API reveals how the operation works; unauthorised changes create an even bigger problem because they interfere with normal workflow. The papers are not security papers, but they still make the API layer look sensitive.

Besada et al. propose a drone-as-a-service architecture with mission planning, resource brokerage, operation support, telemetry, video or data retrieval, and historical logs after the flight [7]. This paper is close to the present topic because it treats the platform as a full service rather than only a routing calculation. The access model is also important. In such a platform, one person could create a mission, a viewer might only watch a feed, and an administrator would need access to logs and system settings. This role separation creates a strong reason to check GraphQL authorisation at query, mutation, field and resolver level.

Bono et al. study UAV fleet path planning and control in bridge management systems. Their work uses digital twin, BIM or BrIM, GIS, virtual waypoints and autonomous trajectories for infrastructure inspection [8]. The bridge inspection context is different from package delivery, but it makes data sensitivity easier to understand. A bridge inspection system stores structural data, exact waypoints, inspection trajectories, camera outputs and GNSS-denied navigation information. A leak of this data is not only a privacy issue. It exposes information about critical infrastructure, so API security becomes part of the system design rather than an optional feature.

Drone fleet data also changes continuously. Rigas et al. formulate drone scheduling for monitoring missions with spatial, temporal and energy constraints, comparing ILP, Greedy and Ant Colony Optimisation methods [9]. Their model uses demand nodes, supply nodes, battery replacement, time slots and scheduling constraints. In a working

system these states appear through live telemetry, planning data or scheduling decisions. The logs need domain fields such as drone ID, mission ID, battery status, operation type and scheduling-related state, not only HTTP method and status code.

Taken together, these operational papers show a recurring pattern. Fleet platforms need to know where drones are, what they are doing, who is responsible for them, and what state the mission is in. These details look ordinary in a management system, but they become sensitive when seen from a security angle. A normal user interface might hide this complexity, while a poorly protected API can expose it directly. The API is treated as part of the operational system, not as a small technical extra.

2.3 Reliability, Availability and Trust

Reliability literature gives a second angle on trust in drone fleets. Fesenko et al. model multi-fleet reliability with reserve drone fleets [10]. Zaitseva et al. review reliability assessment methods for drone swarms, including availability, redundancy, centralised and decentralised control, homogeneous and heterogeneous fleet structures, and importance measures [11]. Their later paper presents Python software for calculating fleet availability from topological properties [12]. Although these papers are not direct cybersecurity sources, they show that a drone fleet must be trusted as a system, not just as a set of individual drones.

Reliability and cybersecurity look separate at first. Reliability papers ask whether the drones and the fleet structure will complete the mission, while cybersecurity asks whether attackers or unauthorised users can disturb the system. In practice, the two questions meet. If the fleet is physically available but the API lets a low-privilege user change mission data, the operation is not trustworthy. The drone hardware still works, but the information guiding the drone becomes doubtful; the same issue appears when telemetry is scraped, hidden or manipulated through the API.

Control architecture is another point that comes from the reliability literature. In centralised control, a supervisor or control unit becomes a major dependency. If that unit fails, the fleet is affected. In decentralised control, the risk is spread out, but coordination and consistency become harder. GraphQL has a similar trade-off at the API level. A single GraphQL gateway makes data access easier, but it also concentrates access control, query validation, logging and monitoring in one place. Since GraphQL normally works through one endpoint, the system has to inspect the content of the query instead of trusting the endpoint alone.

The software tool of Zaitseva et al. is not a security tool, but it frames fleet management as something that can be measured and modelled [12]. That idea is useful for this project. Availability is measured through drone types, redundancy and control style; in the same way, security evaluation should report test results, detection metrics and forensic reconstruction quality. The dissertation should not simply state that the prototype is secure. It has to show what was tested, what was detected, and what evidence remained in the logs.

2.4 GraphQL APIs and the Shift from Endpoints to Fields

The API literature is the second main foundation of the review. REST has been the traditional style for web APIs, while GraphQL has become popular because it allows the client to ask for exact data through a schema. Quiña-Mera et al. describe GraphQL as a query language and execution engine and present a systematic mapping study of 84 primary studies from an initial set of 3,185 [13]. Their review shows that GraphQL adoption is growing, but applied domains still need more empirical study. A drone fleet platform is one such applied domain where GraphQL should be tested beyond simple web examples.

GraphQL solves some REST problems, especially over-fetching and under-fetching. In REST, collecting related data often requires several endpoints, and each endpoint can return more data than the client needs. GraphQL allows the client to request a nested structure in one operation. Brito et al. show this benefit in their migration study, where GraphQL reduced returned JSON fields and bytes in several cases, although it did not always reduce the number of calls [14]. Lawi et al. also compare REST and GraphQL performance, showing that the result is not one-sided because REST performed better in some response-time and throughput measures, while GraphQL showed advantages in CPU and memory use [15].

For drone fleet management, GraphQL has a natural attraction. A dashboard could request drone status together with the latest telemetry and mission details in one operation. An operator may only need mission progress, while a forensic analyst will later need logs for a selected user, drone and time period. The flexibility is useful, but it also means one schema can expose many sensitive paths if authorisation and query controls are weak.

GraphQL's schema is a central concept. It defines the types, fields, queries, mutations, and relationships. In a drone fleet platform the schema could include User, Role, Drone,

Mission, Telemetry, SecurityLog, Alert, BatteryStatus, and Location. The schema helps developers and clients understand the system, but it also acts like a map of the system. If introspection is open, an attacker may learn the available objects and fields. This is not automatically a vulnerability in every environment, but in a sensitive platform it supports reconnaissance. OWASP guidance on GraphQL recommends disabling or controlling introspection when it is not needed, and using query validation and depth or complexity controls [16].

Resolvers are another key point. A resolver is the function that returns data for a GraphQL field. In a simple REST endpoint, developers may place access control at the endpoint level. In GraphQL, this is not enough because a user can request nested fields from related objects. If the top-level query only checks that the user is logged in, but the nested resolver does not check ownership or role, data can leak. For example, a pilot may be allowed to see their own drone telemetry but not telemetry from another fleet. A weak resolver may still return related data if the field is reachable through the graph. This makes field-level and resolver-level authorisation necessary.

Performance and availability risks also appear in GraphQL. Deep nested queries and complex queries can force the server to perform many resolver calls or database operations. This becomes a denial-of-service style problem even without a huge number of HTTP requests. Belhadi et al. focus on automated testing for GraphQL APIs, which supports the idea that GraphQL needs specialised testing because its input space is based on schema and query structure [17]. A normal API test that only checks endpoints may miss the harmful combination of fields, nesting, and arguments.

OWASP API Security Top 10 is useful as a general security baseline [18]. Broken object level authorisation, broken authentication, broken object property level authorisation, unrestricted resource consumption, and excessive data exposure are all relevant to GraphQL drone APIs. GraphQL also needs some specific control points, such as limiting query depth, limiting complexity, preventing alias abuse, securing introspection, validating inputs, applying role checks inside resolvers, and logging the operation with selected fields. Put simply, general API security and GraphQL-specific security have to work together.

2.5 GraphQL Security Risks in a Drone Fleet Context

The API layer in a drone fleet system is better understood as an operational layer, not just a software convenience. It exposes read actions such as drone status, telemetry

history, mission plans, operator details and logs. It also exposes write actions such as creating a mission, updating mission state, assigning drones and acknowledging alerts. A weak read action leads to exposure, while a weak write action changes operational state. For this reason, security testing has to cover both queries and mutations.

Broken access control is one of the main problems. In a GraphQL system, it can happen when a user calls a query or mutation that should be restricted. It can also happen more quietly through nested fields. A user may be allowed to see a mission object but not the full telemetry of every drone connected with that mission. If the schema permits nested telemetry and the resolver does not check the user's permission, the user may extract data indirectly. This makes authorisation more complicated than simple URL endpoint checking.

Excessive data exposure is another likely issue. GraphQL reduces over-fetching when the client is honest because the client can ask for fewer fields. But an attacker can use the same flexibility to ask for many fields. In a drone fleet system, sensitive fields include exact location, route, pilot identity, battery level, mission target, payload type, and security logs. The schema should not expose fields just because they exist in the database. Sensitive fields need permission checks, and some fields usually need to be hidden from the public schema or separated into privileged types.

Query depth and query complexity also need to be tested. A deep query may move from user to missions, from missions to drones, from drones to telemetry, from telemetry to alerts, and from alerts to logs. Even if every field looks valid on its own, the whole query may be expensive. A query complexity attack tries to make the server spend high resources. In a live drone fleet dashboard, this can affect availability because legitimate operators may not receive timely updates. For the proposed security assessment, query depth abuse and query complexity abuse should be included as test scenarios.

Insecure introspection has to be handled carefully. Introspection is useful during development because it lets developers inspect the schema. In a production system with sensitive domain data, uncontrolled introspection often gives attackers a list of types, fields, mutations, and relationships. This can make targeted attacks easier. In the proposed project, introspection can be tested as a scenario by asking what a low-privilege or unauthenticated user can learn from the API schema. The result can then guide the recommended controls.

Unauthorised mutation is more serious than read-only exposure. A query may leak data, but a mutation may change the state of a mission. For example, an attacker could try to

update mission status, reassign a drone, create fake telemetry, or acknowledge an alert. Even if the prototype does not control real drones, the simulated mission state can show the security risk. The test does not need to damage real hardware. It only needs to demonstrate how weak authorisation in the API could affect operational data.

These attack scenarios also connect with forensic logging. A blocked attack has limited investigative value if the log does not show what actually happened. A useful log should show the timestamp, user, role, IP address, operation name, query type, selected fields, target drone or mission, authorisation result, response status, error message, and anomaly score. If the request is a mutation, the log should keep enough context to show what state change was attempted. If the request was blocked, the reason for blocking should also be recorded.

2.6 AI-Based Anomaly Detection for API Behaviour

Artificial intelligence and machine learning appear in the literature in two different ways. One use is operational: RL or other AI methods are used to optimise fleet behaviour, energy usage, connectivity, routing, or decision making. Faraci et al. use reinforcement learning to manage a drone fleet that provides connectivity to IoT devices, taking into account bandwidth demand and wind energy availability [19]. Tchappi et al. discuss explainable agent-based models for remote drone fleet management, where explanations help users understand system behaviour at different levels [20]. These studies show that AI is already being connected with drone fleet control and decision support.

This project uses AI for security rather than flight or route optimisation. Its practical AI component learns from structured GraphQL request logs and flags unusual API behaviour. This narrower focus keeps the work manageable while remaining connected to the reviewed operational AI literature.

A useful starting point is feature design. GraphQL request behaviour can be described using fields such as query type, operation name, query depth, query complexity, number of requested fields, response time, status code, authorisation outcome, failed attempts, user role, target drone ID, target mission ID, source IP, and time of day. These features can be combined with labels such as normal access, broken access-control attempt, telemetry scraping, excessive query depth, query complexity abuse, insecure introspection, and unauthorised mutation. Because public datasets for GraphQL drone-

fleet attacks are not available, a synthetic dataset can be justified if the simulation rules are explained honestly.

Decision Tree is a simple model that is useful for explainability. It may show that high query depth, failed authorisation, and cross-drone access are strong indicators of suspicious activity. Random Forest can improve performance by combining many decision trees and reducing overfitting. Isolation Forest is helpful when the exact attack labels are limited because it detects unusual observations based on how easily they can be separated from normal data. These models are not the newest deep-learning methods, but for a structured log dataset they are reasonable and easier to explain.

Deep learning and reinforcement learning should be discussed in the literature review, but they do not have to be implemented in the practical work. Nandhini and Thinakaran discuss deep reinforcement learning for forensic investigation and security surveillance [21]. Their paper shows that advanced AI is part of the wider security literature, but this dissertation does not need to copy that level of complexity. A smaller MSc project benefits more from explainable models and clear evaluation than from a heavy model that is difficult to justify.

For this practical project, explainability matters more than using the most advanced model name. If a Random Forest flags a request because query complexity, failed access attempts and unusual target resources all rise together, that result is easier to discuss than a black-box score with no explanation. A university project also has to be repeatable. The evaluation should show the dataset fields, the training process, the confusion matrix and one or two example alerts, so the reader can understand how the detection result was produced.

Evaluation should not only report accuracy. If attacks are fewer than normal requests, accuracy can look high even when the model misses important attacks. Precision, recall, F1-score, and confusion matrix are better. Recall is especially important if the aim is to avoid missing suspicious API behaviour. Precision matters because too many false alerts can overload investigators. F1-score balances the two. The model should also be compared against simple rule-based detection, because it is possible that some attacks are easily caught by rules while others need behavioural detection.

AI detection should not be treated as a replacement for secure design. A model detects suspicious patterns after they happen, but access control should still stop unauthorised actions. In the proposed methodology, AI works as a second layer. First, the GraphQL API is tested for controls such as authentication, role-based authorisation, query depth

limits, and mutation restrictions. Then the logs from normal and attack scenarios are used for anomaly detection. Finally, the forensic analysis checks whether the detected incident can be explained from the evidence.

2.7 Digital Forensic Logging and Incident Reconstruction

Digital forensics is needed because security work does not end at detection. After an alert, someone has to investigate what happened. Schmerl et al. discuss explorative visualisation of log data for forensic analysis and IDS signature development [22]. Their paper explains that manual log analysis is time-consuming because a large amount of audit data usually needs to be reviewed. They argue that better visual representation helps investigators recognise relationships between events. The exact 3D tool from their paper does not have to be recreated for this project, but the idea of making logs easier to investigate is very important.

A normal application log might record timestamp, endpoint, status code, and error message. For forensic analysis of a GraphQL drone fleet API, this is not enough. The investigator needs to know the GraphQL operation name, query or mutation type, selected fields, query depth, query complexity, variables used, user identity, user role, source IP, target drone, target mission, response status, authorisation result, and anomaly score. Without these details, a suspicious event may be detected but not explainable.

Timeline reconstruction is a key forensic output because it turns separate log entries into an incident story. A low-privilege user might begin with introspection, then query drone lists many times, then request telemetry outside their role, and finally attempt an unauthorised mission mutation. None of these events should be read in isolation. When the entries are ordered by time and connected with user ID, role, IP address, operation name and target resource, the investigator can explain how the behaviour developed. This is especially useful in GraphQL, where the suspicious part is often inside the query body rather than in the URL.

Attribution is another forensic concern. In a controlled prototype, attribution means linking activity to a user account, role, session or token, IP address, and target resource. It does not prove the real human identity in the outside world, but it supports system-level investigation. If the same user account generates repeated failed authorisation attempts and then performs a high-complexity query, the forensic report can explain that

pattern. If logs do not include user role or authorisation result, the investigation becomes weaker.

Evidence integrity should also be considered. In a real forensic process, logs must be protected from tampering. For the prototype, a simple evidence hash is generated for each log record or for the incident timeline. This does not replace a full chain-of-custody process, but it demonstrates the basic idea of forensic readiness in a way that is realistic for the project scope.

Visualisation is helpful, but it should stay optional. The forensic literature supports visual analysis, although a full dashboard would increase the scope too much. A more suitable approach is to prepare a clear timeline report first. Later, the same timeline could be shown in a simple dashboard if time remains. The priority is not visual polish; it is whether the evidence explains the incident clearly.

This part of the review also identifies a small but important difference between security logging and forensic logging. Security logging often asks whether an event should trigger an alert. Forensic logging asks whether the same event will still make sense later, when the analyst has to explain it in a report. That needs context. A log entry saying 'access denied' is not enough. It should also say which role was used, which field or mutation was requested, what resource was targeted, and how the request fits with earlier activity.

Forensic readiness also links back to the API design. If the API logs only successful requests, failed attacks will disappear from evidence. If it logs only HTTP metadata, GraphQL-specific abuse will be hidden. If it logs raw sensitive data, the log itself may create a privacy risk. Therefore, the logging design must balance detail and safety. It should record the selected field names and identifiers, but not necessarily store complete sensitive telemetry payloads unless required.

2.8 Combined Discussion and Research Gap

When the literature is viewed together, the main pattern becomes visible. Drone fleet studies are strong in routing, scheduling, energy, reliability, digital twin, and mission management. GraphQL studies are strong in schema design, flexible querying, REST comparison, migration benefits, and testing concerns. Cybersecurity guidance explains API risks and GraphQL-specific controls. Forensic literature explains why logs and visual analysis matter for investigation. The problem is that these streams normally stay separate.

The missing part is an integrated view of a drone fleet management API as both an operational control layer and a forensic evidence source. A drone fleet system is not just storing normal business records. It stores live telemetry, route data, mission status, pilot identity, emergency response information, delivery destinations, industrial layouts, or infrastructure inspection waypoints. If a GraphQL API exposes this data, then API security becomes directly connected with operational risk. If the system does not log the right details, then suspicious API behaviour may be difficult to investigate.

The literature gap can be stated in a direct way: current research does not sufficiently combine GraphQL-specific API security assessment, AI-based anomaly detection from API behaviour, and forensic-ready log analysis in drone fleet management systems. Drone papers discuss management, routing, scheduling and reliability. GraphQL papers discuss API flexibility, performance and security. Forensic papers discuss logs and investigation. The missing part is the connection between these areas in one drone fleet case study.

This gap is not only academic; it also affects the methodology. If the problem is integration, the method should also be integrated. A literature-only dissertation would not show whether logs support detection and investigation. A GraphQL prototype without AI would not address behavioural anomaly detection. A machine-learning experiment without drone-fleet context would lose the domain meaning. A forensic discussion without API testing would not show how the evidence is produced. This integrated methodology needs prototype design, attack simulation, log collection, AI detection, and forensic reconstruction.

The scope still needs control. The project should not try to build a real drone platform, a full digital twin, a complete routing optimiser, and a deep-learning security system at the same time. The literature supports all these areas, but they can remain as context. The practical contribution can focus on a simulated GraphQL-based drone fleet management system with enough realism to represent users, roles, drones, missions, telemetry, and logs. This keeps the work achievable while still making it meaningful.

The conceptual framework has four connected layers. The first layer is the drone fleet domain, including drones, missions, telemetry, battery and route-related data. The second layer is the GraphQL API, which exposes this data through queries, mutations, fields and resolvers. The third layer is security monitoring, where logs are collected and anomaly detection is applied. The final layer is forensic reconstruction, where the same

logs are used to explain what happened during an incident. This framework offers a more coherent approach than treating detection and forensics as separate afterthoughts.

This framework also explains the research questions. One question can ask what GraphQL security risks are most relevant in a simulated drone fleet management API. Another can ask whether structured API logs support AI-based anomaly detection. A third can ask whether those logs are useful for forensic reconstruction. These questions are connected, but they are not too broad if the prototype remains simulated and the attack scenarios are clearly selected.

2.9 Critical Comparison of the Main Literature

A critical reading of the drone fleet papers shows that many of them are strong in mathematical or architectural modelling, but they do not fully discuss the security layer that would protect the model in real deployment. Routing and scheduling papers usually assume that the input data is correct and authorised. Demand nodes, supply nodes, battery values, and mission timings are treated as valid inputs to the algorithm. This is normal for optimisation research, but from a security angle it leaves one question open: what happens if the data entering the scheduler is obtained from, or changed through, an insecure API? That question is not the main aim of those papers, but it is important for this dissertation.

The disaster-relief model by Rabta et al. is useful because it deals with situations where drones may deliver urgent items under energy and payload limitations [1]. A strength of this work is that it makes the drone problem realistic by considering recharging stations and priority policies. The limitation for this dissertation is that the model does not explore the software security of the platform that would hold disaster locations, item priorities, and route decisions. In a real emergency system, such information may be sensitive. A weak API could reveal which locations are underserved or which relief items are being transported. This does not reduce the value of the optimisation model, but it gives a reason to add a security layer in future research.

The transportation review by Turgut and Şeker provides a broad overview of drone logistics research [2]. Its strength is that it collects many studies and identifies advantages and challenges in the field. A broad review is useful for understanding trends, but it usually cannot go deeply into one cyber risk. The paper helps justify why drone transport is an active research area, while this project narrows the discussion to

the API layer that supports such transport. In this way, the review gives the general direction and the dissertation adds a security and forensic lens.

The industrial studies by Olivares, Cordova, Córdova, and Olivares are strong because they connect drone fleet management with production planning, battery changes, routing, task selection, and energy efficiency [5], [6]. These papers are practical in the sense that they think about plant demand and material movement. However, they do not ask how a user or system component is authenticated before creating a mission request or changing a route. They also do not ask whether the logs are enough to investigate a wrong mission assignment. In a factory setting, this could be very important, because a small API misuse may delay production or expose layout and inventory information.

The drone-as-a-service paper by Besada et al. comes close to the software architecture of this dissertation [7]. It includes roles, mission definition, resource allocation, telemetry, video or data retrieval, and post-flight logs. Its strength is that it treats drone management as a service platform rather than only an algorithm. The limitation is that security testing and forensic readiness are not the central focus. It mentions historical logs, but it does not define what GraphQL-specific request details should be logged, how suspicious API behaviour could be detected, or how an investigator could reconstruct an API-based misuse event. This creates one of the most direct links to the gap of the present research.

The bridge inspection paper by Bono et al. is useful for another reason: it makes data sensitivity visible [8]. Bridge management systems can include digital twins, GIS information, virtual waypoints and inspection trajectories. A leak of this data exposes critical infrastructure information. Although this paper initially appears to be a path-planning source, it also illustrates why mission and inspection data need access control at the API level.

Reliability papers are important, but their value for cybersecurity is indirect. Fesenko et al. and Zaitseva et al. discuss fleet availability, redundancy and control architecture [10]-[12]. These concepts explain how a fleet continues operation after component failure. They usually do not discuss what happens when API data is accessed or changed by the wrong user. That missing security angle is where this project adds its own direction.

GraphQL papers have their own strengths and gaps. Brito et al. show practical benefits of GraphQL migration, especially reduction in response fields and bytes [14]. Lawi et al. provide a more balanced view by showing that REST can be faster in some metrics

while GraphQL can be efficient in resource utilisation [15]. Quiña-Mera et al. give a systematic mapping of the GraphQL research area and identify the need for stronger empirical studies [13]. These papers are valuable, but they are not specific to drone fleet security. They explain why GraphQL is useful and where it has challenges, while this dissertation places those challenges inside a sensitive cyber-physical domain.

OWASP guidance is necessary because it gives accepted security categories [18], [16]. However, security guidance by itself is not a dissertation methodology. It tells developers what to protect, but it does not automatically show how to evaluate a simulated drone fleet GraphQL API, generate an anomaly dataset, and build a forensic timeline. The proposed research uses OWASP as a foundation rather than the final contribution. The contribution comes from applying these controls and risks to a focused drone fleet case study.

The forensic log visualisation paper by Schmerl et al. is older than many drone papers, but it remains relevant because the log problem has not disappeared [22]. In fact, modern systems produce even more logs. Their argument that manual log analysis is difficult is still true. The difference in this dissertation is that the logs come from GraphQL API activity in a drone fleet prototype. The 3D visualisation approach is not copied directly. Instead, the principle is used: logs should be structured in a way that helps the investigator see relationships between events.

2.10 Limitations in Existing Research

There are several limitations in the existing literature when viewed from the angle of this dissertation. The first limitation is fragmentation. Drone fleet papers study operations, GraphQL papers study API design, AI papers study detection or optimisation, and forensic papers study logs and investigation. Each area is useful, but few studies combine them. This creates a practical difficulty for researchers who want to secure a modern drone fleet platform because the guidance is spread across different fields.

The second limitation is that many drone fleet studies assume trusted inputs. Optimisation models usually take demand, energy, location, and mission parameters as given. Reliability models take drone availability and topology as given. Scheduling models take nodes and time windows as given. These assumptions are reasonable for their research purpose, but real platforms receive inputs through software interfaces. If

those interfaces are not protected, the assumptions may not hold. A corrupted input can affect the mission even if the algorithm is mathematically correct.

The third limitation is lack of GraphQL-specific drone security work. General API security is well discussed, but GraphQL changes how access and data exposure occur. REST has many endpoints, while GraphQL has one endpoint with many possible query shapes. In drone fleet systems, this means one endpoint may provide access to drones, missions, telemetry, users, and logs depending on the query. If the security design is weak, an attacker can move through the graph of data. The literature does not yet provide enough case studies for this exact problem.

The fourth limitation is limited forensic readiness. Some drone platform papers mention historical logs or stored mission data, but they do not treat logs as forensic evidence. A log that is useful for replaying a flight is not always enough for investigating an API attack. For example, a flight log may show where a drone travelled, but it may not show that a low-privilege user sent an unauthorised mutation five minutes before the route changed. Security logs and operational logs need to be connected if the incident is to be investigated properly.

The fifth limitation is explainability in AI security. Advanced models detect anomalies, but the forensic value is weaker if the analyst cannot understand why an event was flagged. This matters in a university project too. A model that gives a high score without explanation is less useful than a simpler model that points towards suspicious features such as query depth, failed authorisation or repeated access to restricted resources.

The sixth limitation is dataset availability. Cybersecurity research often depends on public datasets, but there is no standard dataset for GraphQL API attacks in drone fleet management. This creates a challenge for evaluation. Synthetic data fills the gap only when the generation process is transparent, the labels are explained and the limitations are clearly stated.

The seventh limitation is that many studies focus on either prevention or optimisation, while the investigation phase receives less attention. In security practice, incidents may still happen even when preventive controls exist. Forensic readiness is therefore part of responsible system design. A drone fleet API should not only block obvious abuse; it should also leave enough evidence for later review when something suspicious happens.

These limitations make the research gap more specific. The gap is not simply that 'not enough work exists on drones' or 'not enough work exists on GraphQL'. The gap is the missing connection between the API security layer, behavioural detection, and forensic evidence in a drone fleet management context. This wording is more balanced because it does not ignore existing work. It respects the literature while showing where this project fits.

2.11 Chapter Summary

The literature shows a clear convergence of operational and security concerns. Fleet research demonstrates that missions, energy, routes, reliability, and resources form a connected data environment. Work on modular delivery optimisation, vehicle routing, and implementation feasibility illustrates how strongly this literature prioritises operational planning [23], [24], [25]. GraphQL research explains why flexible field selection is useful in such an environment and why the same flexibility creates risks involving authorisation, resource consumption, schema discovery, and automated testing. Detection research offers methods for behavioural classification, while forensic work shows the need for contextual, chronological, and trustworthy evidence. The remaining gap is practical integration. The next chapters translate that gap into an implemented architecture and an evaluation that separates prevention, detection, and evidential claims.

Broader GraphQL guidance presents advantages such as typed schemas and client-selected data alongside challenges involving performance, security, and maintainability [26]. More recent adoption research shows that type-preserving automation can reduce the effort of introducing GraphQL into existing systems [27]. These advances improve understanding and adoption, but they do not remove the need to connect runtime controls with behavioural detection and preserved incident evidence in an operational domain.

Chapter 3: Research Methodology

3.1 Research Design

The project follows a design-science strategy supported by controlled experimental evaluation. The work did not begin with the final integrated architecture. The first working slice exposed fleet data through GraphQL; role checks, request analysis, evidence capture, detection, and reporting were then added around the same request path. Repeated tests changed several design decisions. Token parsing was tightened, permissions were moved to server-side resolver checks, hash inputs were limited to capture-time fields, and outcome-derived variables were removed from the model feature list. This build-test-revise sequence made design science more appropriate than a purely descriptive threat review.

Evaluation combines quantitative and qualitative evidence. The measured evidence includes automated test outcomes, confusion matrices, precision, recall, F1 score, error rates, ROC-AUC, cross-validation, rule-based results, and evidence-chain counts. The qualitative part asks whether a stored event can actually be explained: who acted, which operation and asset were involved, what the system decided, and whether later model enrichment leaves the captured source evidence intact. This distinction became important when perfect supervised scores appeared; the software pipeline had worked, but the data still required a more sceptical interpretation.

3.2 Artefact and Experimental Environment

The artefact runs locally on Python 3.11 or later and exposes a FastAPI service on the loopback interface. Strawberry provides the GraphQL schema and resolver integration, SQLAlchemy provides object-relational mapping, and SQLite stores operational and security records. Scikit-learn implements the ML pipelines. A static HTML, CSS, and JavaScript dashboard calls the GraphQL endpoint and authenticated export routes. Docker Compose provides a second reproducible execution path with persistent mounts for data, models, and outputs.

The local environment reduces ethical and operational risk. No external target is scanned, no real credentials are used, and no physical drone receives commands. Seeded accounts and synthetic scenarios allow repeated tests without affecting third parties.

This isolation also limits external validity, so the results are presented as proof-of-concept evidence rather than production assurance.

3.3 Dataset Construction

The mixed dataset contains 300 records: 210 normal and 90 anomalous. Normal records represent routine fleet queries and mutations such as listing drones, viewing missions, adding telemetry, and creating missions. Anomalous records represent six controlled operation families: telemetry scraping, introspection, failed login behaviour, mission access, nested query abuse, and unauthorised mission updates. Each row stores behavioural and contextual values needed by the evaluation.

The dataset was intentionally designed so that labels, scenario names, anomaly scores, HTTP status, and authorisation outcomes are not model inputs. These fields would reveal or closely encode the expected outcome and could create target leakage. The ML feature set contains query depth, query complexity, response time, target drone identifier, target mission identifier, role, operation type, and operation name. Numeric features are scaled, categorical features are one-hot encoded, and unknown categories are tolerated by the preprocessing pipeline.

Table 3.1 Dataset composition

Class / family	Records	Purpose
Normal behaviour	210	Routine fleet, mission, and telemetry activity
allTelemetry	20	High-volume telemetry access pattern
IntrospectionQuery	19	Schema-discovery behaviour
login	18	Repeated invalid authentication pattern
missionById	12	Direct mission access pattern
updateMissionStatus	11	Unauthorised mission manipulation
nestedTelemetry	10	Deep or expensive nested request

3.4 Security Assessment Scenarios

The assessment combines negative and positive cases. Negative cases verify that anonymous access, forged or expired tokens, excessive depth, non-administrator introspection, unauthorised evidence downloads, and viewer mutations are rejected. Positive cases verify that authorised users can complete permitted fleet operations and

that administrators and analysts can access security and forensic functions. This matters because a control that blocks everything is secure only in a trivial sense; the system must preserve legitimate functionality.

Attack simulation is deliberately separated from live observed activity. Simulations create labelled records with an explicit reason, while middleware-generated audit records describe actual local requests. A source-classification function marks this distinction for evidence presentation. The separation reduces the risk that a demonstration silently treats invented events as independent evidence of an attack.

3.5 Detection Methods

The rule-based detector assigns transparent weights to depth, complexity, failed or suspicious authorisation, viewer mutations, introspection behaviour, scraping indicators, and blocked status codes. Scores at or above 0.55 are labelled anomalous. The approach is easy to explain and maps directly to policy, but its thresholds may fail when suspicious behaviour is weak across several fields.

Five machine-learning models provide different comparison points. Decision Tree offers interpretable splits. Random Forest and Extra Trees combine multiple trees and reduce dependence on a single partition of the data. Logistic Regression provides a linear probabilistic baseline. Isolation Forest is trained only on normal training records and represents unsupervised novelty detection [28]. Random Forest follows the ensemble principle described by Breiman [29]. Supervised models use class weighting where appropriate, a stratified 70/30 hold-out, and five-fold stratified validation.

3.6 Evaluation Metrics

Table 3.2 Evaluation metrics

Metric	Interpretation in this study
Accuracy	Proportion of all records classified correctly; potentially misleading with class imbalance.
Precision	Proportion of predicted anomalies that are genuine anomalies; low precision means alert overload.
Recall	Proportion of genuine anomalies detected; low recall means attacks are missed.
F1 score	Harmonic balance between precision and recall.
False-positive rate	Normal records incorrectly escalated; important for analyst workload.

Metric	Interpretation in this study
False-negative rate	Anomalies missed by the detector; important for security coverage.
ROC-AUC	Ranking quality across thresholds, interpreted with the other metrics.
Evidence integrity rate	Proportion of stored records whose content and chain relationships verify.

3.7 Reproducibility, Ethics, and Validity

Reproducibility is supported through version-pinned dependencies, a maintained PowerShell launcher, container configuration, seeded accounts, scripted model training, stored prediction files, and automated tests. A fixed random state of 42 is used for dataset splitting and model estimators. The project records model outputs rather than relying on screenshots alone.

Ethically, all attacks are simulated within a local system owned for the project. The dataset does not contain personal or operational records from real people or fleets. Demo credentials are unsuitable for deployment and are clearly documented as seeded local accounts. The token secret must be changed before assessment outside the local environment. The prototype avoids claiming legal admissibility: an application hash chain supports tamper detection, but formal chain of custody also requires controlled acquisition, identity, storage, procedural documentation, and independent oversight. NIST log-management guidance emphasises the organisational process around log infrastructure and maintenance [30]. The interpretation of model results also follows the NIST AI Risk Management Framework principle that validity, reliability, transparency, and accountability must be considered together rather than reducing responsible AI to a single accuracy value [31].

Internal validity is threatened by synthetic labels, shared generation rules, and the possibility that response time or identifiers act as strong proxies for the class. Construct validity is limited because the scenarios represent selected GraphQL abuse patterns rather than the full threat landscape. External validity is limited by one local application, SQLite, a small dataset, and no real network or fleet. Conclusion validity is limited by only 90 hold-out records. These threats motivate the unseen-operation-family experiment and cautious interpretation of perfect supervised scores.

3.8 Methodological Rationale and Alternatives

An exclusively quantitative design was rejected because metric values alone could not show whether the captured evidence was useful for investigation or whether the implemented controls matched the conceptual risk. A purely qualitative case study was also insufficient because the detector and enforcement mechanisms required repeatable tests. Combining artefact construction with controlled measurement allowed the research to examine both technical behaviour and the meaning of the resulting evidence.

The study did not use interviews or surveys. User perceptions could be valuable for dashboard usability, but they would introduce recruitment, consent, and sampling requirements without directly answering the principal security question. The selected method instead treats the software, dataset, logs, tests, and generated artifacts as the units of analysis. This choice keeps the project feasible and reproducible, although it means that claims about analyst usability remain untested.

Alternative validation strategies were considered. A time-based split would better represent deployment drift, but the synthetic dataset did not contain a sufficiently long and independent chronology. Cross-project validation would be stronger, but no equivalent public dataset contained the same GraphQL and fleet fields. The leave-one-operation-family-out procedure was therefore added as an intermediate test. It removes one attack family from training while retaining the common feature space, providing more evidence than a random split without pretending to be external validation.

The methodology also distinguishes verification from validation. Verification asks whether the implementation behaves as specified: tokens expire, role checks reject forbidden actions, hashes recalculate, and exports require authorisation. Validation asks whether the design addresses the research problem: whether the evidence supports reconstruction and whether detection can distinguish meaningful behaviour. Passing verification tests is necessary but does not settle the validity questions, which are addressed through critical interpretation and stated limitations.

Chapter 4: System Requirements and Design

4.1 Requirements

Requirements were derived from the research gap and from authoritative GraphQL and API-security guidance. The GraphQL specification defines a hierarchical, strongly typed, client-specified, and introspective interface [32]. OWASP identifies authorisation, resource consumption, insecure configuration, and sensitive business-flow abuse as major API concerns [18]. Its GraphQL guidance recommends depth and amount limiting, cost analysis, rate limiting, resolver-level access checks, and restricted introspection [16]. The design converts these recommendations into testable controls rather than presenting them only as advice.

Table 4.1 Functional and security requirements

ID	Requirement	Implemented response
FR1	Manage users, drones, missions, and telemetry	GraphQL queries/mutations and dashboard views
FR2	Support investigation and reporting	Evidence filters, timeline, CSV export, PDF report
SR1	Authenticate users	One-hour HMAC-signed bearer token
SR2	Restrict functions by role	ADMIN, OPERATOR, ANALYST, VIEWER resolver checks
SR3	Limit abusive GraphQL documents	AST depth, complexity, operation-count, and batch checks
SR4	Reduce automated abuse	Separate login and general API rate limits
SR5	Preserve forensic context	Automatic structured audit record for every GraphQL POST
SR6	Detect evidence alteration	Per-record SHA-256 content hash and linked chain hash
SR7	Support behavioural detection	Rules and five ML models with leakage-controlled features

4.2 Architecture

The architecture has six logical layers. The presentation layer is a static dashboard that manages login state, sends GraphQL documents, displays fleet records, starts controlled simulations, runs detection, builds timelines, verifies evidence, and downloads reports. The API layer uses FastAPI and Strawberry. The security middleware sits before

GraphQL execution and parses the incoming document into an abstract syntax tree. Resolver-level access functions then apply role rules to accepted documents.

The service layer contains modules for users, fleet, missions, telemetry, attack simulation, explainable detection, ML detection, security logging, forensics, and reporting. SQLAlchemy maps these services to SQLite tables. The evidence layer stores audit context and linked hashes. Finally, the artifact layer stores trained models, prediction CSV files, explainability results, and reports. This separation keeps operational records distinct from generated evaluation artifacts while allowing the dashboard to coordinate them.

Implemented GraphQL security and evidence architecture

Code-derived view of the request, enforcement, persistence and analysis path

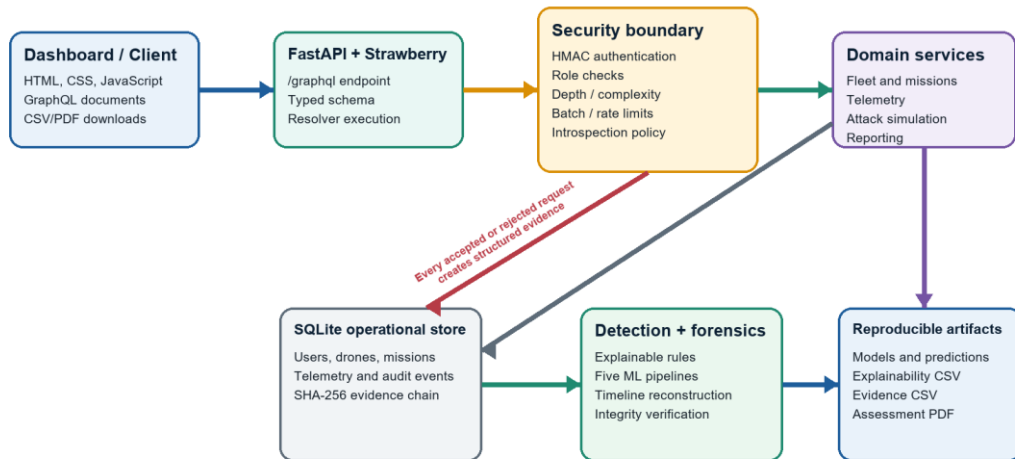


Figure 4.1 Implemented GraphQL security and evidence architecture (author-generated from the project code)

Figure 4.1 makes the main design decision visible: prevention and evidence capture are not separate afterthoughts. The security boundary examines a request before resolver execution, while both accepted and rejected outcomes are passed to the evidence store. Detection and forensic reconstruction then operate on the same recorded sequence rather than on a disconnected demonstration dataset.

4.3 Authentication and Authorisation Design

Successful login returns a compact local token containing user identifier, username, role, issue time, and expiry time. The payload is base64url encoded and signed with HMAC-SHA256. Validation compares signatures using a constant-time function, checks the token format, accepts only known roles, and rejects expired tokens. This

design is intentionally smaller than a complete OAuth or OpenID Connect deployment, but it demonstrates integrity and expiry controls suitable for the local artefact.

Passwords created by the current application use PBKDF2-HMAC-SHA256 with a unique 16-byte salt and 210,000 iterations. A compatibility branch accepts legacy SHA-256 hashes in earlier seeded databases. That branch is useful for continuity but should be removed by migrating all records in a real deployment. Role checks are repeated in protected resolvers and REST-style export routes so that hiding a dashboard button is never treated as authorisation.

4.4 GraphQL Abuse-Control Design

The middleware analyses the GraphQL document before resolver execution. It calculates maximum depth, estimates lightweight complexity, counts operations, expands fragments, and detects introspection. The configured default limits are depth 8, complexity 80, at most 3 operations, 300 general GraphQL requests per IP per 60 seconds, and 10 login requests per IP per 60 seconds. Introspection is restricted to administrators. A rejected request produces a structured error and an anomaly evidence record with the reason and calculated values.

This order is important. Expensive or unauthorised documents should be rejected before business logic touches the database. At the same time, the rejected request should not disappear from the forensic record. Accepted requests are timed around GraphQL execution and then logged with actor, IP address, raw query, operation type and name, depth, complexity, status, and authorisation state.

4.5 Evidence and Forensic Design

Each security record stores the time, user and role, source address, operation, selected query, structural measures, response time, status, authorisation result, optional drone or mission identifiers, anomaly score, label, reason, evidence hash, previous hash, and chain hash. The content hash covers stable source-evidence fields and deliberately excludes later ML enrichment. Consequently, running detection can update a label or explanatory reason without pretending that the original event fields changed.

The chain hash links the previous chain value to the current evidence hash. Verification recalculates content hashes and chain relationships in order. This supports detection of modification, deletion, or reordering within the stored sequence, subject to the limits of local keyless hashing and database control. The forensic report orders matching

evidence newest first for presentation while calculating totals over the complete matching set. Suspected users and affected assets are derived from all matching records, not only the displayed page.

4.6 Attack and Detection Design

Table 4.2 Controlled attack scenarios

Scenario	Security question	Expected evidence
Broken access control	Can a low-privilege actor read protected mission data?	Denied actor, role, operation, target and risk reason
Deep query abuse	Can nested selections exceed the depth policy?	Calculated depth, blocked status and query
Introspection probe	Can a non-admin enumerate the schema?	Introspection flag and administrator-only rejection
Telemetry scraping	Can repeated broad access indicate extraction?	Operation family, complexity and scraping reason
Mission tampering	Can a viewer perform a privileged mutation?	Role, mutation, target mission and denial
Failed login burst	Can repeated credential failures be identified?	Anonymous/login pattern and failure evidence

4.7 User Interface Design

The dashboard is organised around the assessment workflow rather than around database tables alone. Overview presents fleet and security totals. Fleet Operations creates legitimate baseline activity. Attack Laboratory creates controlled anomalous evidence. AI/ML Detection compares rules and models. Forensic Analysis reconstructs suspicious activity. Evidence Logs filters and exports records. Buttons carry role metadata and are disabled for unauthorised users, while server checks remain authoritative.

The complete-demo action sequences attack simulation, rule-based analysis, Random Forest classification, timeline reconstruction, integrity verification, and refresh. This reduces demonstration error but does not replace the ability to run each stage separately. Live security alerts use an authenticated newline-delimited JSON stream available only to administrators and analysts.

4.8 Trust Boundaries and Failure Handling

The design identifies the browser, API process, database, model files, and exported artifacts as separate trust boundaries. Browser state is untrusted because a user can change JavaScript, enable disabled buttons, or construct requests manually. Every permission is therefore repeated on the server. The API process is trusted to apply policy but receives hostile input. The database is trusted for persistence but not assumed to be independently immutable, which is why verification detects internal inconsistency without claiming an external chain of custody.

Failure handling aims to preserve a useful record without exposing excessive detail. Malformed, deep, costly, batched, rate-limited, and restricted-introspection requests produce short client messages and richer internal evidence. Invalid tokens fail closed. Missing model files produce a controlled application error rather than silently switching algorithms. Database sessions are scoped through service calls, and the test suite checks that invalid actions do not bypass permission through alternative routes.

Chapter 5: Implementation

5.1 Technology Stack

Table 5.1 Technology stack

Component	Technology	Reason
Web service	FastAPI 0.141.1	Typed Python service, middleware, streaming, and file responses
GraphQL	Strawberry GraphQL 0.324.0	Python schema types and FastAPI integration
Persistence	SQLAlchemy 2.0.52 / SQLite	Inspectable relational model and local reproducibility
ML	scikit-learn 1.9.0	Comparable supervised and unsupervised pipelines
Frontend	HTML, CSS, JavaScript	No build dependency; direct GraphQL demonstration
Deployment	PowerShell and Docker Compose	Windows assessment workflow and container reproducibility
Testing	pytest 9.1.1 / FastAPI TestClient	Automated API, security, integrity, and artifact checks

5.2 Operational GraphQL Schema

The schema exposes user, drone, mission, telemetry, security-log, detection, integrity, and forensic types. Queries retrieve operational records, security metrics, filtered evidence, integrity status, and forensic reports. Mutations authenticate users; create, update, and delete operational records; add telemetry; simulate controlled attacks; run rule-based detection; train models; and apply a selected model to current database evidence.

Service functions separate persistence logic from GraphQL resolver definitions. This keeps access control visible at the API boundary while avoiding direct SQL in the schema. Telemetry and timeline queries return newest records first. Aggregate security and forensic totals are calculated independently of presentation limits so a dashboard page does not silently become the total population.

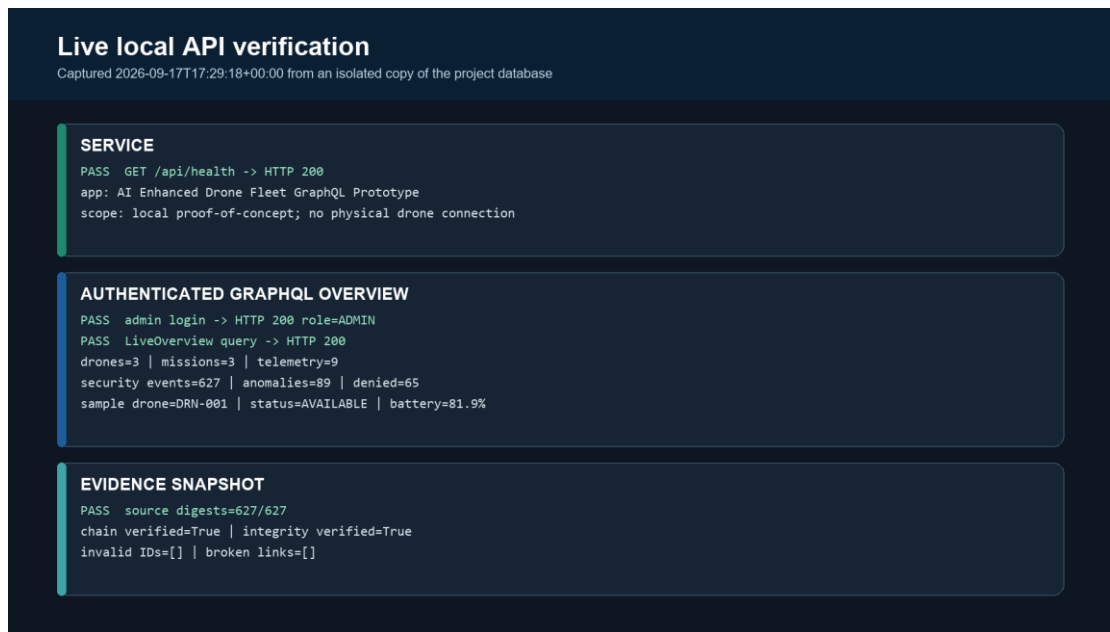


Figure 5.1 Live local API verification from an isolated copy of the project database

The capture in Figure 5.1 was generated by the running application rather than prepared as a mock interface. It confirms the seeded fleet totals, an authenticated administrator query, and the evidence state returned by the same GraphQL schema used by the dashboard. Tokens were redacted from the saved capture.

5.3 Request Security Implementation

The GraphQL middleware handles only POST requests to the GraphQL route. It reads JSON payloads, supports detection of batched documents, analyses the raw query, selects a separate login or general rate-limit bucket, and parses the bearer token. Rejection priority is rate limit, syntax, excessive operations, restricted introspection, excessive depth, and excessive complexity. Rejections are returned with HTTP 400, 403, or 429 as appropriate, and rate-limit responses include a retry header.

The AST analyser is safer than estimating depth from braces because it understands fragments, inline fragments, operations, and introspection fields. Tests specifically verify fragment expansion. Lightweight complexity is not a complete execution-cost model, but it is deterministic and suitable for demonstrating pre-execution policy. A production service should calibrate field-specific resolver costs, pagination limits, database query plans, and timeouts.

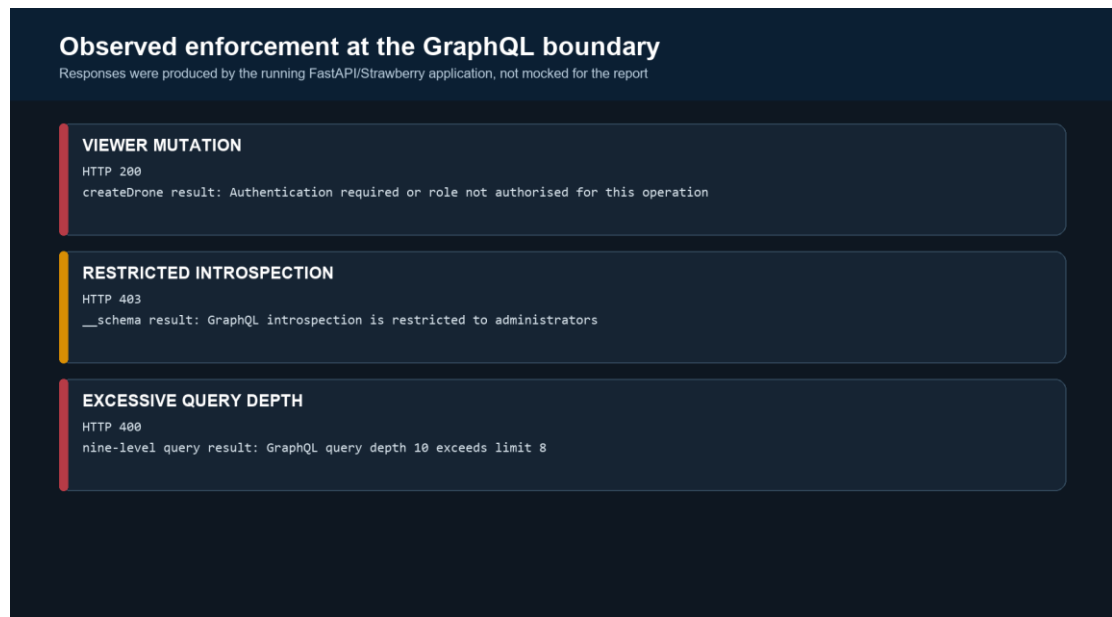


Figure 5.2 Observed GraphQL security enforcement responses

Figure 5.2 also records a protocol detail that became clear during testing. The viewer mutation reached GraphQL execution and returned HTTP 200 with a GraphQL error, whereas middleware rejections for introspection and excessive depth returned HTTP 403 and 400. In every case the requested action failed. Reporting both transport status and GraphQL error content avoids treating HTTP 200 alone as evidence of authorisation success.

5.4 Audit and Evidence Implementation

Accepted operations are measured with a high-resolution performance timer. The middleware extracts an explicit operation name where available and uses a neutral anonymous name otherwise. It then creates an audit record after the response. Rejected requests are recorded before returning. This produces visibility across successful, unauthenticated, and policy-blocked traffic.

The evidence export route regenerates a CSV from the database and requires administrator or analyst access. The assessment-report route uses the same role restriction and produces a PDF. Integrity verification returns total and valid counts, invalid record identifiers, broken chain identifiers, and Boolean content/chain results. During ML enrichment, the source evidence hash remains stable; an automated test compares hashes before and after a changed risk explanation.

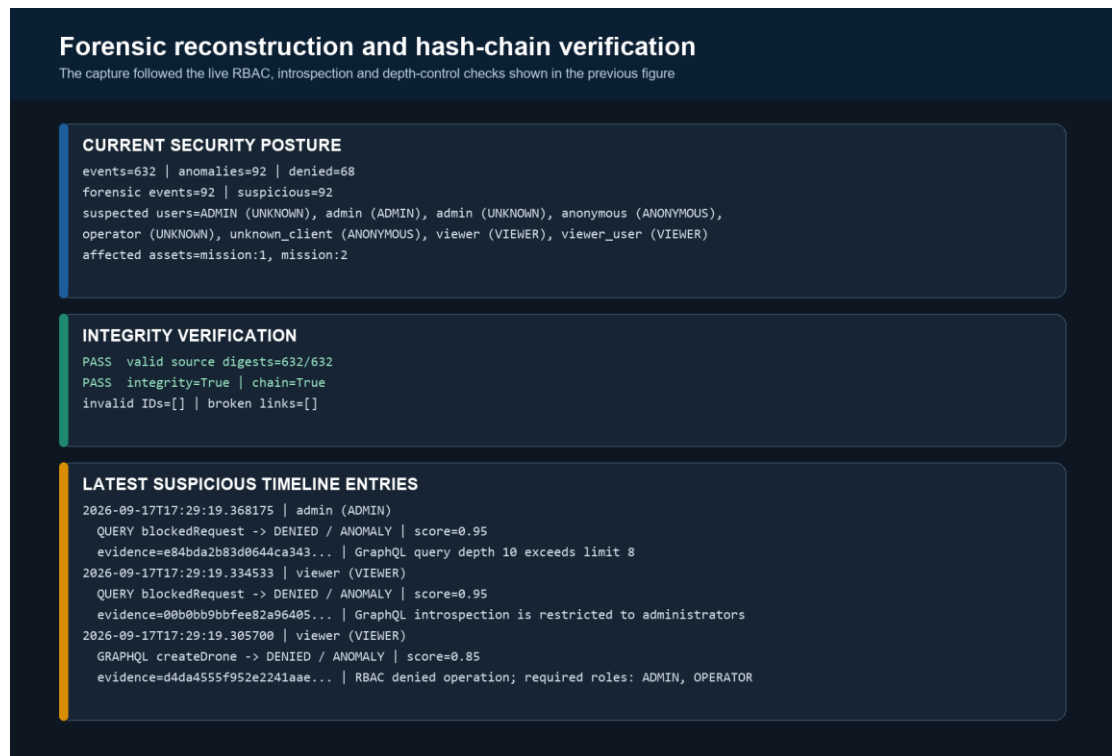


Figure 5.3 Forensic reconstruction and evidence-chain verification after the live control checks

The main evaluation snapshot reported later in Chapter 6 contained 617 valid records. For the revised report, that database was copied before any new request was sent. The isolated capture then recorded logins, the overview query, the denied viewer mutation, restricted introspection, excessive depth, and verification requests, reaching 632 valid records in Figure 5.3. The different totals are therefore expected and provide direct evidence that the system records its own assessment activity without changing the original database.

5.5 Rule-Based and ML Implementation

The rule engine provides a direct explanation for each score. Depth at least 5 adds 0.25, complexity at least 18 adds 0.25, failed or suspicious authorisation adds 0.30, a viewer mutation adds 0.25, introspection adds 0.25, scraping adds 0.25, and a 401 or 403 status adds 0.20. Scores are capped at 1.0. Pre-labelled simulation records receive a minimum score of 0.70 so a later run does not erase the intended scenario label.

ML preprocessing uses a ColumnTransformer. Numeric fields are converted safely and standardised. Role, operation type, and operation name are upper-cased and one-hot encoded. The Decision Tree depth is limited to 6. Random Forest uses 150 trees with maximum depth 8 and balanced class weights. Extra Trees uses 200 estimators with

maximum depth 10. Logistic Regression uses balanced weights and up to 1,500 iterations. Isolation Forest uses 200 estimators and trains only on normal examples.

Each model is saved with joblib, and test predictions are written to CSV. Supervised feature importance is reported from model coefficients or impurity importance, while model-agnostic permutation importance measures the reduction in F1 when a feature is shuffled. The Random Forest is also retrained while holding out each anomalous operation family. This tests a stronger question than a random split: whether the model can flag a family that was absent from its training attacks.

5.6 Forensics and Reporting Implementation

Forensic reconstruction filters security records by label when requested, calculates complete totals, identifies suspicious actors and referenced assets, and sorts events by timestamp and record identifier. Each timeline entry combines actor, role, action, target, outcome, anomaly score, evidence hash, and investigative note. This design supports the practical questions who acted, what was requested, which asset was involved, what the system decided, and why the event was considered risky.

The reporting module converts evidence and results into examiner-friendly artifacts. CSV supports independent analysis in spreadsheet or forensic tools. PDF provides a stable assessment summary. The dashboard offers direct downloads, but access remains checked at the server. The live alert stream starts at the latest known record and emits new anomaly events, with heartbeat messages when no new anomaly is available.

5.7 Deployment and Reproducibility

On Windows, the maintained launcher creates or reuses the virtual environment, installs pinned dependencies, optionally resets data, trains missing models, and starts Uvicorn at 127.0.0.1:8000. Normal startup preserves the database. A separate reset switch makes destructive rebuilding explicit. Docker Compose builds the image, supplies environment configuration, mounts persistent data/model/output directories, and exposes the same port.

The environment file controls database location, debug mode, GraphQL limits, rate windows, maximum operations, introspection policy, token lifetime, and secret. The repository includes an example rather than a deployment secret. This makes security-relevant defaults visible while reinforcing that the local default must not be used for external deployment.

Chapter 6: Evaluation and Discussion

6.1 Evaluation Procedure

Evaluation was conducted at three levels. First, the automated pytest suite exercised the application in-process with isolated audit cleanup. Second, live HTTP and GraphQL checks were sent to the running service to verify the deployed path, static assets, seeded roles, exports, and reports. Third, dataset scripts evaluated rule-based and ML detection and wrote prediction artifacts for inspection. The evidence database was not reset during the final live checks, allowing chain verification over the accumulated demonstration records.

Table 6.1 Verification summary

Area	Result	Interpretation
Automated suite	24/24 passed	Core API, security, evidence, ordering, exports, and container artifacts passed
Python compilation	Passed	Application, scripts, and tests compiled without syntax errors
Frontend delivery	Passed	Dashboard and all referenced local CSS/JS assets returned successfully
RBAC	Passed	Anonymous access, viewer mutation, and non-admin introspection were rejected
Exports	Passed	Anonymous export rejected; authorised CSV and PDF generated
Evidence integrity	617/617 valid	Original evaluation snapshot; later isolated capture verified 632/632
HTML structure	Passed	No duplicate IDs and no missing referenced local assets

```
Automated regression test evidence
Fresh run against an isolated copy of the demonstration database

TEST SESSION
$ python -m pytest -q
..... [100%]
===== warnings summary =====
tests/test_smoke.py::test_admin_can_edit_and_delete_unlinked_fleet_records
tests/test_smoke.py::test_admin_can_edit_and_delete_unlinked_fleet_records
D:\UWTSO\Term3\Project\drone_graphql_fleet_man\.venv\Lib\site-
packages\sqlalchemy\sql\schem.py:3627: DeprecationWarning: datetime.datetime.utcnow() is
deprecated and scheduled for removal in a future version. Use timezone-aware objects to
represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    return util.wrap_callable(lambda ctx: fn(), fn) # type: ignore
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
24 passed, 2 warnings in 8.46s
```

Figure 6.1 Fresh automated regression-test run against an isolated database copy

The fresh run in Figure 6.1 completed 24 tests in 8.46 seconds. The two warnings came from SQLAlchemy calling the deprecated naive UTC helper during timestamp defaults; they did not fail a test, but they identify a concrete maintenance task: move persisted timestamps to timezone-aware UTC values before the dependency removes that compatibility path.

6.2 Functional and Security Results

The automated suite covered core functional checks for health, protected queries, valid login, authorised fleet access, and create/update/delete workflows. Security tests addressed forged and expired tokens, depth blocking, restricted introspection, AST fragment handling, protected alert streaming, and protected evidence and report downloads. Evidence-oriented tests checked uncapped security metrics, source classification, stable hashes during ML enrichment, leakage-controlled feature lists, linked evidence order, and newest-first forensic and telemetry output. The suite also confirmed five-model availability, dashboard content, and reproducible container files.

Live testing confirmed three drones, three missions, and nine telemetry records in the preserved demonstration database at the time of assessment. The dashboard returned HTTP 200, and each linked stylesheet and JavaScript file was available. Admin and viewer accounts authenticated with their seeded local credentials. An administrator could retrieve dashboard data and schema introspection, while a viewer was blocked from creating a drone and from introspection. Invalid passwords produced a GraphQL

error. CSV evidence and the PDF report were inaccessible anonymously and valid when requested with an administrator token.

These results answer RQ1 and RQ2 at proof-of-concept scale. The relevant GraphQL risks are not limited to malformed input; valid documents can be unsafe because of field access, nesting, operation count, repetition, or schema discovery. The prototype demonstrates that structural analysis, identity, role policy, and forensic capture can share one request path. The passing tests do not establish absence of vulnerabilities, but they provide traceable evidence for the controls claimed in the report.

6.3 Rule-Based Detection Results

The standalone rule-based dataset test processed all 300 records. It produced 60 true positives, 210 true negatives, no false positives, and 30 false negatives. Accuracy was 0.900, precision 1.000, recall 0.667, and F1 0.800. The result illustrates a useful trade-off: the rule set generated no false alerts on this dataset, but it missed one-third of anomalies. In operational terms, an analyst would receive a clean alert queue but could overlook behaviours that do not cross the fixed score threshold.

This outcome supports the decision to retain rules as an explanation and policy layer rather than use them as the only detector. The rule features are understandable and can justify a blocked request. However, detection logic that depends on explicit words such as introspection or scraping may not generalise to renamed operations or distributed behaviour. Temporal aggregation and rate-derived features would likely improve recall.

6.4 Machine-Learning Results

Table 6.2 Machine-learning evaluation results

Model	Accuracy	Precision	Recall	F1	ROC-AUC	5-fold F1
Decision Tree	1.000	1.000	1.000	1.000	1.000	0.9943 ± 0.0114
Random Forest	1.000	1.000	1.000	1.000	1.000	0.9943 ± 0.0114
Logistic Regression	1.000	1.000	1.000	1.000	1.000	1.0000 ± 0.0000
Extra Trees	1.000	1.000	1.000	1.000	1.000	1.0000 ± 0.0000
Isolation Forest	0.3444	0.3140	1.000	0.4779	0.9148	0.4852 ± 0.0049

Detection performance on the 300-record synthetic dataset

The contrast is more informative than the perfect supervised hold-out scores alone

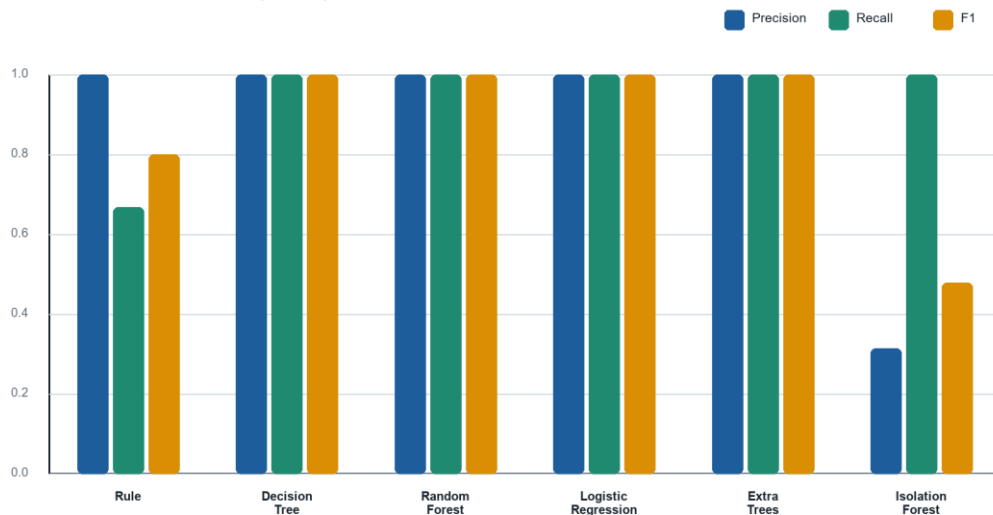


Figure 6.2 Detection precision, recall, and F1 comparison on the 300-record dataset

All four supervised models classified the 90-record hold-out set perfectly, with a confusion matrix of 63 true negatives and 27 true positives. Cross-validation remained close to perfect. This is encouraging as a software-pipeline result: preprocessing, training, persistence, prediction, and metric reporting operate consistently, and outcome-derived fields are absent from `FEATURE_COLUMNS`. It is not sufficient evidence that real attacks will be detected perfectly.

Several factors can explain the strong scores. Synthetic normal and anomalous records were generated with clear behavioural differences. Response time, target identifiers, role, complexity, and operation names may separate the classes even when explicit labels are removed. A random split allows closely related examples from the same operation family to appear in training and test sets. Permutation analysis reinforces this concern: response time caused the largest F1 decrease for Decision Tree and Logistic Regression, while most Random Forest feature permutations caused little change on the already easy hold-out set.

Isolation Forest presents the opposite result. It detected every anomaly but incorrectly labelled 59 of 63 normal records, giving a false-positive rate of 0.9365. Its ROC-AUC of 0.9148 indicates useful ranking information, but the default decision threshold is unsuitable for this dataset. This distinction is important: a detector may rank risk well while producing an unusable binary alert volume. Threshold calibration, contamination estimates, and a larger normal baseline are required before unsupervised deployment.

Confusion matrices expose different operational trade-offs

Rows are actual classes; columns are predicted classes

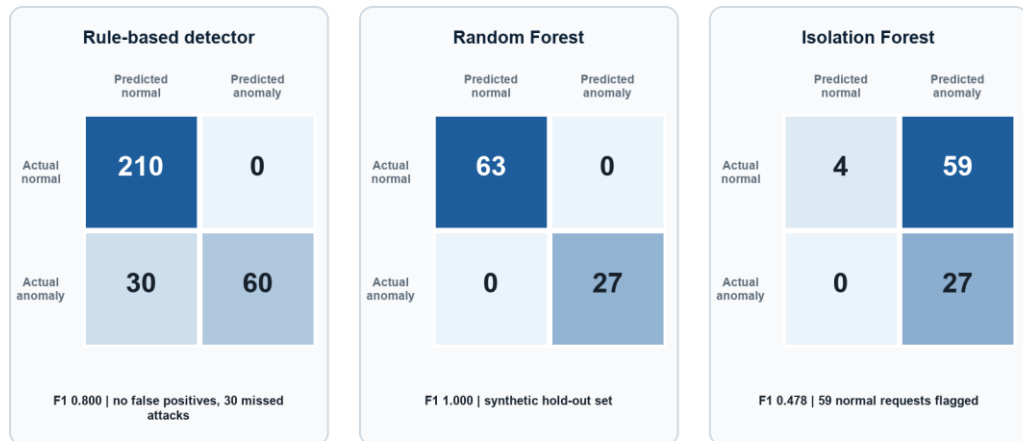


Figure 6.3 Confusion matrices for three contrasting detection approaches

Figure 6.3 changed the practical interpretation more than the headline accuracy values did. The rule set was quiet but incomplete, Random Forest separated this synthetic hold-out perfectly, and Isolation Forest would have overwhelmed an analyst with false alerts. The figure therefore supports a layered design rather than selection of one apparently best model.

6.5 Unseen Operation-Family Evaluation

Table 6.3 Leave-one-operation-family-out results

Held-out family	Records	Detected	Recall	Mean anomaly probability
ALLTELEMETRY	20	20	1.0000	0.9083
INTROSPECTIONQUERY	19	19	1.0000	0.9586
LOGIN	18	17	0.9444	0.6467
MISSIONBYID	12	12	1.0000	0.6350
NESTEDTELEMETRY	10	10	1.0000	0.9798
UPDATEMISSIONSTATUS	11	11	1.0000	0.5958

Holding out each anomalous operation family is stricter than a random split because the Random Forest cannot memorise that family's attack examples. Recall remained 1.0 for five families and 0.9444 for login, where 17 of 18 records were detected. Mean probabilities were lower for updateMissionStatus and missionById than for

introspection or nested telemetry. The model therefore appears to use shared behavioural signals, but confidence varies by family.

This test reduces, but does not remove, the generalisation concern. Normal examples and feature-generation logic remain from the same synthetic environment. The held-out family may still share response-time or role distributions with other attacks. Independent logs from another implementation would provide a more meaningful external test.

Leave-one-operation-family-out Random Forest evaluation

Each attack family is excluded from training and then used as the test set

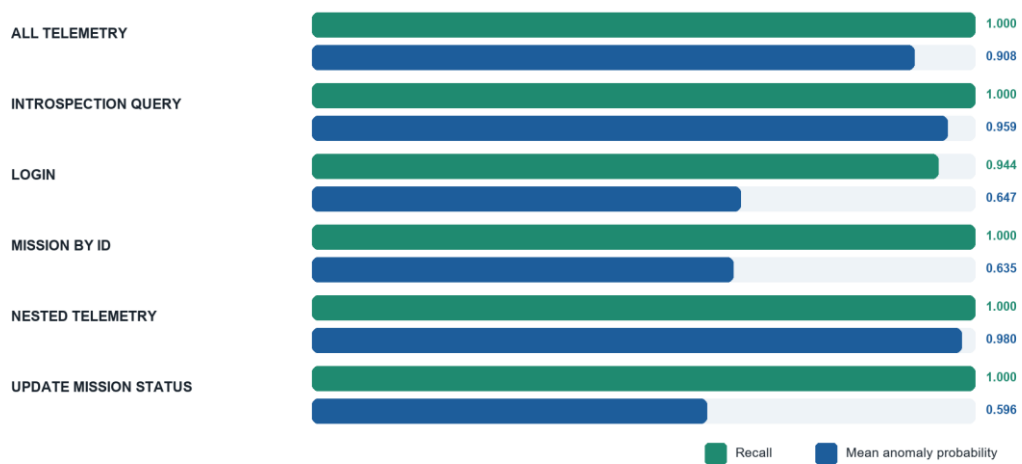


Figure 6.4 Leave-one-operation-family-out Random Forest evaluation

The visual comparison in Figure 6.4 shows why recall alone is not enough. Five held-out families achieved complete recall, but their mean probabilities ranged from 0.5958 to 0.9798. Login produced both the only missed record and a comparatively low mean probability. A deployment threshold that appears safe for nested telemetry could therefore behave differently for an unseen authentication pattern.

6.6 Forensic Evaluation

At the original evaluation snapshot, integrity verification reported 617 valid records out of 617, with no invalid content hashes or broken chain links. The later isolated capture verified 632 out of 632 records, as shown in Figure 5.3. These results confirm that the implemented verifier and each captured database state agree. They do not prove that a database was never replaced as a whole, because the application controls both storage and verification. Stronger assurance would require signed checkpoints, protected key storage, append-only infrastructure, or an external timestamp service.

The forensic timeline answered the intended investigative questions for available data. It identified actors and roles, classified operations, referenced drone or mission assets where present, showed authorisation and anomaly outcomes, and displayed evidence hashes. Totals were not capped by the visual timeline limit. CSV export supported independent examination, while the PDF report provided a stable summary. These features answer RQ4 within the stated local scope and align with the need for organised log-management processes described by NIST [30].

6.7 Difference from Previous Research

Table 6.4 Comparison between previous research and this project

Research area	Typical focus in earlier work	Difference in this project
Drone fleet management	Routing, scheduling, reliability, energy, or service architecture	Uses fleet data as the protected operational context for API security and forensics
GraphQL research	Adoption, performance, schema use, migration, or automated testing	Enforces depth, complexity, batching, rate, RBAC, and introspection policies in one demonstrator
Anomaly detection	Model accuracy on network or log datasets	Compares rules and five models using GraphQL behavioural features and explicit leakage controls
Digital forensics	Log visualisation, investigation frameworks, or evidence preservation	Captures API evidence at request time, links hashes, reconstructs timelines, and exports artifacts
Integrated studies	Limited end-to-end connection across all four areas	Connects prevention, detection, evidence integrity, and reconstruction around the same GraphQL events

The major difference is architectural and evidential. The project does not simply apply a classifier to a drone dataset, nor does it only list GraphQL vulnerabilities. It implements a request lifecycle in which controls and evidence are coupled. A rejected query becomes an investigative object; a permitted query provides a behavioural baseline; a model prediction enriches but does not rewrite source evidence; and a timeline can be verified and exported.

This integration addresses the core gap identified in Chapter 2. It also exposes tensions that separate studies can overlook. Strong prevention reduces the number of successful malicious examples available for detection. Simulated labels make evaluation repeatable but risk unrealistic separation. Mutable enrichment improves interpretation but must not invalidate source hashes. A dashboard improves usability but cannot be trusted to

enforce permission. Treating these tensions explicitly is a substantive contribution of the artefact.

6.8 Limitations

The most important limitation is dataset realism. Three hundred synthetic records cannot represent the diversity, drift, noise, user behaviour, and adversarial adaptation of a deployed fleet. Perfect supervised scores are therefore a prompt for scrutiny, not a production claim. The rule-based detector's 0.667 recall and Isolation Forest's high false-positive rate provide more realistic evidence of trade-offs.

The system is local and uses SQLite. In-memory rate-limit buckets are not shared across processes and reset on restart. HMAC tokens lack revocation, refresh, issuer, and audience controls. The legacy password branch should be removed after migration. GraphQL complexity is an estimate rather than resolver-specific cost. The evidence chain is stored beside the records and has no external trust anchor. IP capture does not account for trusted proxies. The dashboard received static and API-contract testing, but automated visual browser interaction was unavailable during the final test session.

The evaluation also lacks comparison with commercial API gateways, SIEM platforms, or alternative GraphQL security libraries. No real users completed a usability study. No penetration test was conducted against an internet-facing deployment. These omissions are consistent with the safe MSc scope but limit generalisation and publication claims.

6.9 Integrated Discussion by Research Question

For RQ1, the evidence indicates that the most relevant risks arise from the interaction between GraphQL flexibility and operational sensitivity. A drone record is not dangerous merely because it is represented as a graph; risk appears when nested traversal reveals linked telemetry, when identifiers enable unauthorised object access, when mutations change mission state, or when expensive selection sets consume resources. GraphQL-specific controls must therefore be combined with conventional authentication, object-level authorisation, input validation, and rate management. The implemented scenarios cover this combination more effectively than a checklist that treats every GraphQL feature as automatically vulnerable.

For RQ2, the middleware demonstrates a practical integration point. Structural checks occur before execution, resolver permissions protect accepted documents, and both outcomes reach the same evidence service. This reduces the visibility gap between

security and forensics. Nevertheless, coupling every request to synchronous database logging may create a performance or availability dependency at scale. A production implementation should consider a durable queue, back-pressure, protected audit storage, and an explicit policy for what happens when evidence persistence fails.

For RQ3, no single detector is best under every criterion. Rules provide direct reasons and zero false positives in the dataset, but their recall is limited. Supervised models achieve excellent internal separation but depend on labelled examples and require external validation. Isolation Forest catches all attacks but overwhelms the analyst with false alerts. A defensible operational design would use layered decisions: deterministic controls for prohibited behaviour, a calibrated supervised risk score for known patterns, novelty scoring for investigation rather than automatic blocking, and human review for high-impact actions.

For RQ4, the stored fields are sufficient to create a coherent local narrative of actor, request, target, result, risk, and integrity state. The hash chain detects inconsistency across the recorded sequence and the exports support examination outside the dashboard. The design is therefore forensic-ready in the limited sense of intentional evidence capture and verification. It is not a complete forensic process because custody, independent time, host integrity, retention authority, and legal procedure extend beyond the application.

Taken together, the research questions show why the integrated approach matters. Prevention without evidence makes blocked activity difficult to study. Detection without enforced access control leaves the system exposed while waiting for a score. Evidence without behavioural context creates a large but weak audit trail. Machine learning without critical validation creates confidence that the data may not justify. The prototype combines these parts and, equally importantly, makes their limitations visible.

Chapter 7: Conclusions and Recommendations

7.1 Conclusions

This project investigated how GraphQL security assessment, behavioural anomaly detection, and forensic analysis can be integrated in a simulated drone fleet platform. The completed artefact demonstrates that a typed and flexible GraphQL interface can support operational fleet workflows while enforcing authentication, role permissions, query-shape limits, rate limits, and restricted introspection. It also demonstrates that both accepted and rejected requests can be captured as structured evidence and connected to detection and timeline functions.

RQ1 was answered by mapping drone fleet data to GraphQL-specific risks: broken object and function authorisation, excessive field access, deep or complex queries, batching, schema discovery, authentication abuse, and sensitive operational extraction. RQ2 was answered through middleware and resolver controls that share an automatic evidence service. RQ3 was answered through comparative evaluation: transparent rules were precise but missed 30 anomalies; supervised models separated the synthetic dataset perfectly; Isolation Forest detected anomalies but generated excessive false positives. RQ4 was answered through structured timelines, CSV/PDF artifacts, and successful integrity verification of the original 617-record evaluation snapshot and the later 632-record isolated capture.

The main conclusion is not that AI solves GraphQL security. Detection adds value only when combined with enforced controls, appropriate features, critical metric interpretation, and evidence that remains trustworthy after enrichment. The integrated design is the project's main contribution and the strongest basis for future publication.

7.2 Recommendations

- Collect a substantially larger, time-ordered dataset from multiple GraphQL implementations and preserve an independent external test set.
- Calibrate model thresholds against analyst workload and operational risk rather than accepting library defaults.
- Add temporal features such as requests per actor, repeated target access, burst rate, and changes from each user's baseline.

- Replace local HMAC tokens with a mature identity provider and support revocation, rotation, issuer, audience, and multi-factor authentication.
- Move rate limiting and evidence storage to shared, production-capable services when scaling beyond one process.
- Anchor evidence-chain checkpoints externally or sign them with protected keys, while documenting acquisition and custody procedures.
- Use field-specific GraphQL cost models, pagination limits, resolver timeouts, and database query monitoring.
- Conduct independent security testing, usability evaluation, and replication before making operational or legal claims.

7.3 Future Publication Direction

A publishable paper should narrow the contribution to a clear empirical question, such as whether leakage-controlled GraphQL behavioural features generalise across operation families or whether linked application evidence improves incident reconstruction. The next study should preregister dataset rules, publish anonymised data and code, include at least one independent implementation, compare against strong baselines, and report confidence intervals. The current prototype can provide the experimental platform, but publication claims should be based on new external evidence rather than the perfect synthetic hold-out result.

Chapter 8: Reflection

The project did not begin with a fixed API-security question. The original idea covered drones and AI too broadly. Building the first working fleet-data flow showed that the useful research question sat at the API boundary: what access should be allowed, and what evidence remains after each request? Adding authentication and role checks then affected the dashboard, GraphQL resolvers, downloads, live alerts, tests, and the meaning of the audit log. Connecting these parts took longer than adding isolated features, but it made the final demonstration easier to follow from the incoming request to the stored evidence record.

Evidence hashing caused the most important design correction. At first, treating every stored field as immutable appeared simple. It became clear that this would make later analytical enrichment look like tampering because risk labels and explanations can change when another detector is run. The final design hashes capture-time fields, chains those hashes in record order, and includes a regression test showing that ML enrichment does not alter the original hash. This reduced the scope of the forensic claim, but made it more defensible.

The machine-learning results also changed how the evaluation was interpreted. Perfect hold-out scores initially appeared to be the strongest outcome. Checking feature construction and operation-family overlap showed why that interpretation was too confident. Outcome fields were excluded, cross-validation and permutation checks were added, and Random Forest was retested with one attack family removed at a time. Isolation Forest's 59 false positives were kept in the report because they showed the analyst-workload problem that an accuracy-only discussion would miss.

A separate issue emerged during live capture. A viewer's forbidden mutation returned HTTP 200 with a GraphQL error, while middleware decisions for introspection and depth returned HTTP 403 and 400. Checking only the transport status would have led to the wrong conclusion. The capture was run against a copied database and increased the evidence count from the earlier snapshot to 632 valid records. This showed that verification activity becomes part of the audit trail without altering the original assessment database.

Time management improved only after the work was divided into testable slices: operational GraphQL data, enforcement, evidence capture, detection, reconstruction,

and reporting. If the project were repeated, more time would be reserved for an independent traffic dataset, browser-level usability tests, external replication, and structured threat modelling before coding. The final artefact demonstrates feasibility and integration. It does not prove production security, legal admissibility, or universal machine-learning performance, and recognising this boundary became an important part of the research process.

References

- [1] B. Rabta, C. Wankmüller, and G. Reiner, “A drone fleet model for last-mile distribution in disaster relief operations,” *International Journal of Disaster Risk Reduction*, vol. 28, pp. 107–112, 2018.
- [2] M. Turgut and B. Şeker, “Drone Technology in Transportation Management: A Systematic Review and Framework for Future Research,” *Journal of Aviation*, vol. 7, no. 2, pp. 251–261, 2023.
- [3] A. Jazairy, E. Persson, M. Brho, R. von Haartman, and P. Hilletoft, “Drones in last-mile delivery: a systematic literature review from a logistics management perspective,” *The International Journal of Logistics Management*, vol. 36, no. 7, pp. 1–62, 2025, doi: 10.1108/IJLM-04-2023-0149.
- [4] J. Hažík, M. Dekan, P. Beňo, and F. Duchoň, “Fleet Management System for an Industry Environment,” *Journal of Robotics and Control*, vol. 3, no. 6, pp. 779–789, 2022.
- [5] V. Olivares and F. Cordova, “Design of Drone Fleet Management Model in A Production System of Customized Products,” in *Proc. 6th International Conference on Computers Communications and Control*, 2016, pp. 165–172.
- [6] F. Córdova and V. Olivares, “Fleet of drones in a customized and flexible production system,” in *Proc. 2024 IEEE International Conference on Automation/XXVI Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, 2024, doi: 10.1109/ICA-ACCA62622.2024.10766835.
- [7] J. A. Besada, A. M. Bernardos, L. Bergesio, D. Vaquero, I. Campaña, and J. R. Casar, “Drones-as-a-service: A management architecture to provide mission planning, resource brokerage and operation support for fleets of drones,” in *Proc. UNAGI 2019*, pp. 931–936.
- [8] A. Bono, L. D’Alfonso, G. Fedele, A. Filice, and E. Natalizio, “Path Planning and Control of a UAV Fleet in Bridge Management Systems,” *Remote Sensing*, vol. 14, no. 8, Art. no. 1858, 2022.
- [9] E. S. Rigas, P. Kolios, M. Mavrovouniotis, and G. Ellinas, “Scheduling a Fleet of Drones for Monitoring Missions With Spatial, Temporal, and Energy Constraints,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 15133–15145, 2022.
- [10] H. Fesenko, V. Kharchenko, and E. Zaitseva, “Evaluating Reliability of a Multi-fleet with a Reserve Drone Fleet: an Approach and Basic Model,” in *Proc. International Conference on Information and Digital Technologies*, 2019.
- [11] E. Zaitseva, V. Levashenko, R. Mukhamediev, N. Brnzei, A. Kovalenko, and A. Symagulov, “Review of Reliability Assessment Methods of Drone Swarm (Fleet) and a New Importance Evaluation Based Method of Drone Swarm Structure Analysis,” *Mathematics*, vol. 11, no. 11, Art. no. 2551, 2023.
- [12] E. Zaitseva, V. Levashenko, G. Kamalova, and M. Kasak, “Software for Reliability Evaluation of Drone Swarm/Fleet,” in *Proc. 9th International Conference on System Reliability and Safety*, pp. 82–86, 2025.
- [13] A. Quiña-Mera, P. Fernandez, J. M. García, and A. Ruiz-Cortés, “GraphQL: A Systematic Mapping Study,” *ACM Computing Surveys*, vol. 55, no. 10, Art. no. 202, 2023, doi: 10.1145/3561818.
- [14] G. Brito, T. Mombach, and M. T. Valente, “Migrating to GraphQL: A Practical Assessment,” in *Proc. IEEE SANER*, pp. 140–150, 2019.
- [15] A. Lawi, B. L. E. Panggabean, and T. Yoshida, “Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System,” *Computers*, vol. 10, no. 11, Art. no. 138, 2021.
- [16] OWASP Cheat Sheet Series, “GraphQL Cheat Sheet.” [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html. [Accessed: Sep. 17, 2026].
- [17] A. Belhadi, M. Zhang, and A. Arcuri, “Evolutionary-based Automated Testing for GraphQL APIs,” in *Proc. GECCO 2022 Companion*, pp. 778–781, 2022, doi: 10.1145/3520304.3528952.
- [18] OWASP, “OWASP API Security Top 10 – 2023,” 2023. [Online]. Available: <https://api-security.owasp.org/editions/2023/en/0x11-t10/>. [Accessed: Sep. 17, 2026].
- [19] G. Faraci, A. Raciti, S. A. Rizzo, and G. Schembra, “Green wireless power transfer system for a drone fleet managed by reinforcement learning in smart industry,” *Applied Energy*, vol. 259, Art. no. 114204, 2020.

- [20] I. Tchappi, J. E. N. Mboula, A. Najjar, Y. Mualla, and S. Galland, "A Decentralized Multilevel Agent Based Explainable Model for Fleet Management of Remote Drones," *Procedia Computer Science*, vol. 203, pp. 181–188, 2022.
- [21] T. J. Nandhini and K. Thinakaran, "Optimizing Forensic Investigation and Security Surveillance with Deep Reinforcement Learning Techniques," in *Proc. International Conference on Data Science, Agents, and Artificial Intelligence*, IEEE, 2023.
- [22] S. Schmerl, M. Vogel, R. Rietz, and H. König, "Explorative Visualization of Log Data to Support Forensic Analysis and Signature Development," in *Proc. Fifth International Workshop on Systematic Approaches to Digital Forensic Engineering*, pp. 109–120, 2010.
- [23] J. Lee, "Optimization of a Modular Drone Delivery System," in *Proc. 2017 Annual IEEE International Systems Conference (SysCon)*, pp. 1–8, 2017, doi: 10.1109/SYSCON.2017.7934790.
- [24] S. A. Sukarno, "Approximation methods to vehicle routing problem for a drone fleet management," Ph.D. dissertation, Université Polytechnique Hauts-de-France, France, 2019.
- [25] T. Benarbia and K. Kyamakya, "A Literature Review of Drone-Based Package Delivery Logistics Systems and Their Implementation Feasibility," *Sustainability*, vol. 14, no. 1, Art. no. 360, 2022.
- [26] I. Papadhopulli and H. Paci, "GraphQL: A Comprehensive Analysis of Its Advantages, Challenges, and Best Practices in Modern API Development," *The Eurasia Proceedings of Science, Technology, Engineering and Mathematics*, vol. 32, pp. 230–237, 2024.
- [27] S. Amareen, A. Rahman, S. Rahaman, and A. Bosu, "GraphQLify: Automated and Type Safety-Preserving GraphQL API Adoption," *Proceedings of the ACM on Software Engineering*, vol. 3, no. FSE, Art. no. FSE014, pp. 299–320, 2026, doi: 10.1145/3797141.
- [28] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *Proc. Eighth IEEE International Conference on Data Mining*, pp. 413–422, 2008.
- [29] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.
- [30] K. Kent and M. Souppaya, *Guide to Computer Security Log Management*, NIST Special Publication 800-92, Sep. 2006, doi: 10.6028/NIST.SP.800-92.
- [31] E. Tabassi, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, NIST AI 100-1, Jan. 2023, doi: 10.6028/NIST.AI.100-1.
- [32] GraphQL Foundation, "GraphQL Specification, September 2025 Edition," 2025. [Online]. Available: <https://spec.graphql.org/September2025/>. [Accessed: Sep. 17, 2026].

Appendix A: Principal GraphQL Security Tests

- Anonymous protected query is rejected.
- Valid administrator token permits fleet queries.
- Forged and expired tokens are rejected.
- Excessive query depth is blocked before resolver execution.
- Viewer mutation and non-administrator introspection are rejected.
- Evidence and PDF downloads require administrator or analyst access.
- AST analysis expands fragments and identifies introspection.
- Evidence hashes remain stable when ML annotations change.
- Chain links verify in chronological record order.
- Forensic and telemetry output is sorted newest first.

On 20 September 2026, the smoke-test suite was rerun against a temporary SQLite database. All 24 tests passed in 21.65 seconds. The 16 non-failing warnings were SQLAlchemy datetime.utcnow deprecations. Live HTTP and GraphQL checks on the same isolated instance produced the outcomes shown in Figure A.1. The frontend check confirms HTML delivery, not browser rendering or usability.



Figure A.1 Fresh isolated validation summary from actual pytest and live API responses (20 September 2026)

Appendix B: Default Security Configuration

Table B.1 Default local configuration

Setting	Default
GRAPHQL_MAX_DEPTH	8
GRAPHQL_MAX_COMPLEXITY	80
GRAPHQL_RATE_LIMIT	300 requests / 60 seconds / IP
GRAPHQL_LOGIN_RATE_LIMIT	10 requests / 60 seconds / IP
GRAPHQL_MAX_OPERATIONS	3
GRAPHQL_ADMIN_INTROSPECTION_ONLY	True
TOKEN_LIFETIME_SECONDS	3600

Appendix C: Reproduction Commands

Windows launcher: `.\START_HERE_WINDOWS.ps1`

Clean demonstration build: `.\START_HERE_WINDOWS.ps1 -ResetData`

Automated tests: `.\.venv\Scripts\python.exe -m pytest -q`

Dataset rule evaluation: `.\.venv\Scripts\python.exe scripts\test_dataset_rule_based.py`

Model training: `.\.venv\Scripts\python.exe scripts\train_ml_models.py`

Container deployment: Copy-Item `.env.example` `.env` followed by `docker compose up --build`

Appendix D: Word Count Statement

Declared word count for Chapters 1-8: 14681 words.

This count includes chapter headings and table text within Chapters 1-8. It excludes preliminary pages, references, and appendices.

Appendix E: Real-Time Interface and Testing Screenshots

The following screenshots were captured by the author from the locally running Aegis Flight prototype on 21 September 2026. They provide interface-level evidence of the dashboard, controlled attack laboratory, automated demonstration workflow, machine-learning workspace, forensic reconstruction, and evidence-log review. Counts shown in the interface reflect the accumulated local demonstration database at the time of capture and are not substitutes for the controlled evaluation metrics reported in Chapter 6.

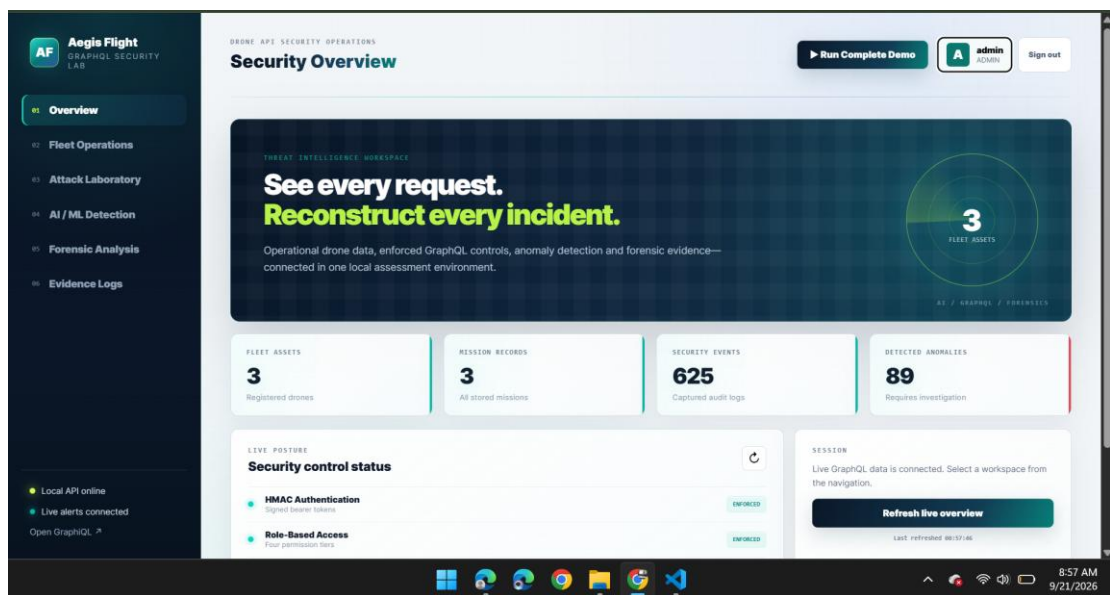


Figure E.1 Authenticated Aegis Flight security overview with live fleet and security counters

The authenticated overview connects operational status with the security posture of the local prototype. At capture time, the interface reported three fleet assets, three mission records, 625 captured security events, and 89 detected anomalies; these values describe the accumulated demonstration database rather than a fixed benchmark dataset.

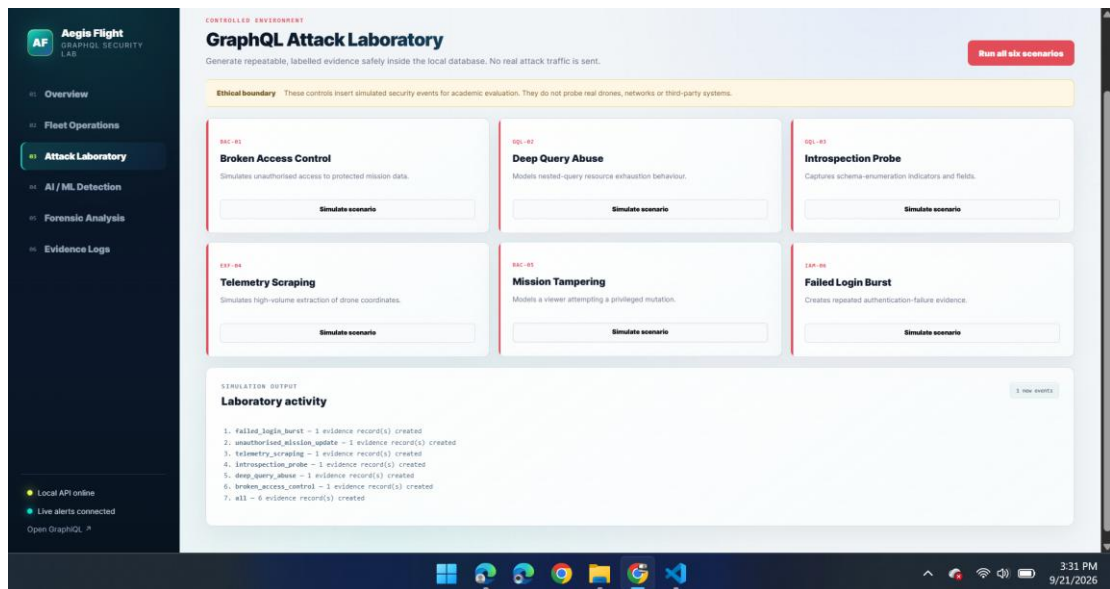


Figure E.2 Controlled GraphQL Attack Laboratory after execution of all six simulation scenarios

The laboratory records one labelled event for each controlled scenario: broken access control, deep-query abuse, introspection probing, telemetry scraping, mission tampering, and failed-login activity. The interface explicitly states the ethical boundary: the scenarios create local academic evidence and do not probe real drones or third-party systems.

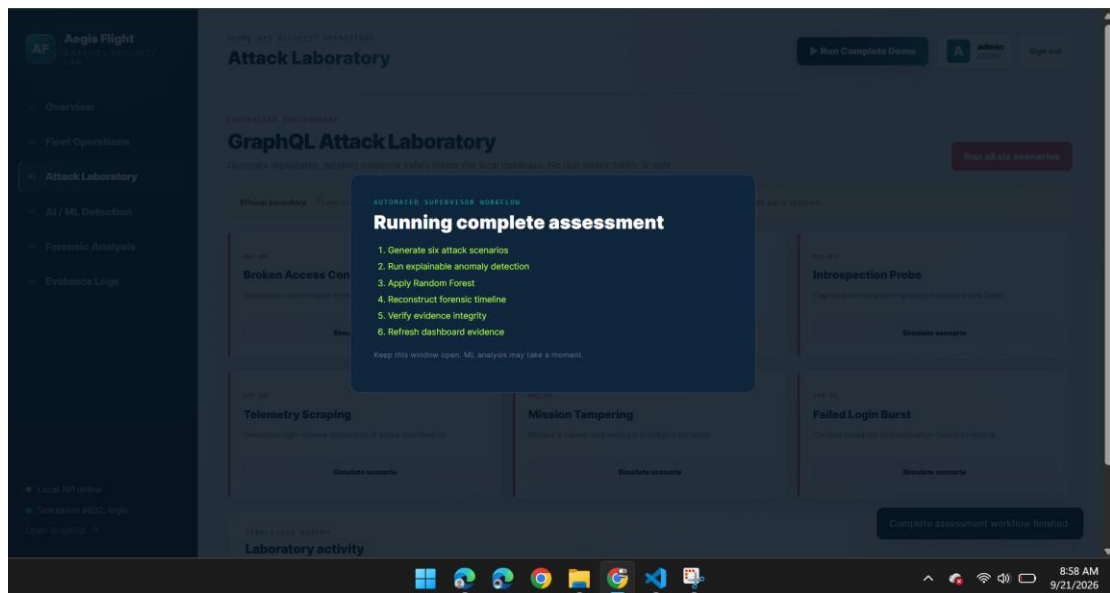


Figure E.3 Automated complete-assessment workflow used for the supervisor demonstration

The complete-assessment control runs the six demonstration stages in a fixed order. The capture shows all stages marked complete and the completion notification, providing interface-level evidence of the integrated workflow rather than evidence of an attack on an external system.

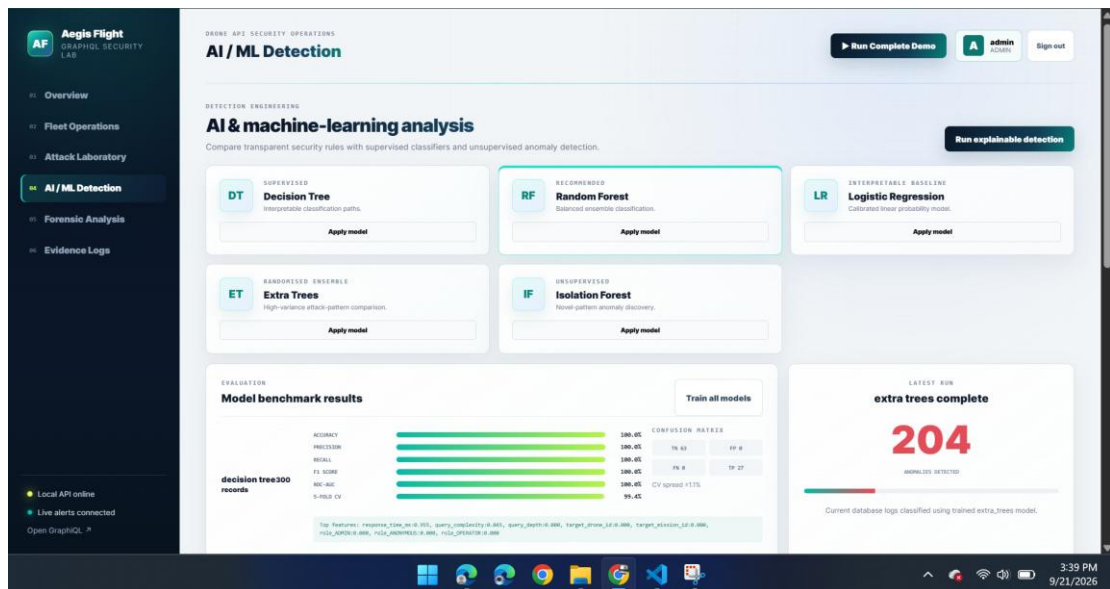


Figure E.4 AI/ML detection workspace after Extra Trees classification of accumulated local logs

Although the source filename refers to Random Forest, the visible latest-run panel states 'extra trees complete'. The 204 value therefore represents Extra Trees classifications over the accumulated local database logs; it is not the Random Forest hold-out score and should not be interpreted as the number of anomalies in the 300-record evaluation dataset.

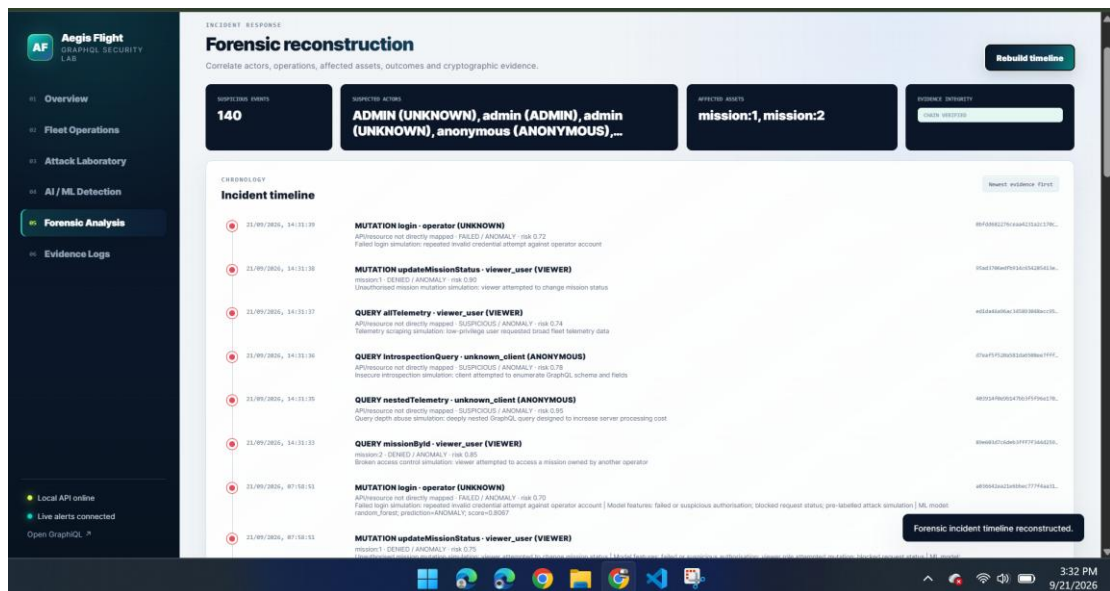


Figure E.5 Forensic reconstruction with verified evidence chain and chronological incident records

The forensic view correlates timestamp, operation, actor, role, target, outcome, risk score, explanatory note, and evidence hash. The displayed total reflects the accumulated local demonstration history, while the chain-verified indicator confirms the state of the stored evidence at the time of capture.

Angis Flight
GRAPHQL SECURITY LAB

01 Overview
02 Fleet Operations
03 Attack Laboratory
04 AI / ML Detection
05 Forensic Analysis
06 **Evidence Logs**

Local API online
Live alerts connected
Open GraphQL

00000 API SECURITY OPERATIONS
Evidence Logs

FORENSIC EVIDENCE STORE
GraphQL security logs
Search captured requests, verify the evidence chain and export examiner-ready artifacts.

Verify chain Export CSV Download PDF report

Search actor, operation or risk reason... ANOMALY Refresh

TIMESTAMP	ACTOR / ROLE	OPERATION	DEPTH / COST	AUTH	RISK	CLASSIFICATION	EVIDENCE HASH
21/08/2026, 14:38:47	anonymous ANONYMOUS 1270:0:1	blockedRequest QUERY	0 / 0	DENIED	0.95	ANOMALY	3f13d03f706a0708a8a407c2d78b0704...
21/08/2026, 14:34:49	anonymous ANONYMOUS 1270:0:1	blockedRequest QUERY	15 / 361	DENIED	0.95	ANOMALY	c2108b7a3a8b0228b2f4a2c208a17f3a2d...
21/08/2026, 14:34:48	anonymous ANONYMOUS 1270:0:1	blockedRequest QUERY	15 / 365	DENIED	0.95	ANOMALY	5a98b76a45512b70a7a5a3a3b4a12af4a4...
21/08/2026, 14:34:13	admin ADMIN 1270:0:1	Model MUTATION	2 / 10	ALLOWED	0.80	ANOMALY	4a13a028a70a0d70a42b203a1e11303a...
21/08/2026, 14:34:11	admin ADMIN 1270:0:1	Model MUTATION	2 / 10	ALLOWED	0.81	ANOMALY	63181c128a0a1082f708b8a3a3a7f03a...
21/08/2026, 14:34:10	admin ADMIN 1270:0:1	Model MUTATION	2 / 10	ALLOWED	0.88	ANOMALY	2a8f48a2c77c1a3a8f02a8a0a0a7a3a...
21/08/2026, 14:34:08	admin ADMIN 1270:0:1	Model MUTATION	2 / 10	ALLOWED	0.81	ANOMALY	c128a028a70a0d70a42b203a1e11303a...
21/08/2026, 14:32:42	admin ADMIN 1270:0:1	Forensics QUERY	3 / 29	ALLOWED	0.88	ANOMALY	50a8a0a70a7f0a3a20a70a20a0a7a3a...
21/08/2026, 14:32:38	admin ADMIN 1270:0:1	Forensics QUERY	3 / 29	ALLOWED	0.88	ANOMALY	3a1a5a0a70a7f0a3a20a70a20a0a7a3a...
21/08/2026, 14:31:39	operator OPERATOR 1270:0:1	login MUTATION	2 / 5	FAILED	0.94	ANOMALY	8a1a5a0a70a7f0a3a20a70a20a0a7a3a...

3:39 PM
9/21/2026

Figure E.6 Evidence-log interface filtered to anomalous GraphQL activity

The evidence table exposes the fields required for review and export, including authorization outcome, query depth and cost, risk score, classification, and hash. Hashes are shortened in the interface for readability; the complete values remain available in the evidence export and database. This capture shows anomaly filtering, while chain verification is evidenced separately in Figures E.5 and A.1.