SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

Chainbreaker: A Graph-Based Framework For Flow-Level Persistent Advanced Threat Detection, Attack Chain Reconstruction, And Forensic Investigation

[1] M.R. Padmapriya, [2] Dr. I. Bremnavas, [3] Vijayakumar Adaikalam, [4] Gopinath. P.G., [5]Nanthini K, [6] J Puspha

[1] Department of CSE (Data Science), Dayananda Sagar College of Engineering, Bengaluru, padmapriya-csds@dayanandasagar.edu
[2] Department of Computer Science & IT, Jain (Deemed-to-be) University, Bengaluru, ibrem.navas@jainuniversity.ac.in
[3]Professor, Presidency School of Artificial intelligence and Advanced Computing Presidency University, Bangalore, India. vijaykumar.adaickalam@presidencyuniversity.in
[4] Department of ECE, R.M.K. Engineering College, Kavaraipettai, Tamil Nadu, India.gopuice@gmail.com
[5] Department of Computer Science and Applications, Christ Academy Institute for Advanced Studies, Bengaluru, Karnataka, India. nanthinksg@gmail.com
[6] Department of Computer Science & IT, Jain (Deemed-to-be) University, Bengaluru, j.pushpa@jainuniversity.ac.in

Abstract — Advanced Persistent Threat (APT) detection remains difficult due to the multi-phase nature of these attacks and the inability to accurately infer them from individual network flows. Current flow-based classification methods can classify traffic individually, but cannot track the context of the attack. To solve this problem, the paper introduces a graph-based APT detection and forensics system called ChainBreaker. Network flow analysis is done using XGBoost model for attacks, and Isolation Forest for anomalies. Detection is assigned to one of the nine MITRE ATT&CK kill chain phases, and then each phase is incrementally represented as live Neo4j property graph that allows incremental building of attack progression by interconnected AttackEvent and KillChainStage objects. Meanwhile, the Apache Spark temporal analytics layer detects coordinated activities such as password spraying and distributed flooding that are not visible in individual flow analysis. In contrast to traditional security architectures that use separate log stores and correlation engines, the proposed architecture maintains a unified graph representation to detect, correlate, and investigate attacks and incidents through graph traversal. The effectiveness of this model was measured using the CICIoT2023 dataset, which consisted of 34 attack types and over 5.3 million network flows. The results showed that validation accuracy of 99.59% and an ROC-AUC score of 0.9999 were achieved using the XGBoost algorithm, but this approach allowed for a complete graph-based construction of the attacks.

Index Terms— Advanced Persistent Threats, Attack Graph, Cyber Kill Chain, Graph-Based Intrusion Detection, MITRE ATT&CK, Neo4j, Network Intrusion Detection System, XGBoost, Isolation Forest, CICIoT2023, Apache Spark, Temporal Aggregation, Forensic Analysis, Kill Chain Interruption.

1. INTRODUCTION

The rapid adoption of cloud computing, IoT, Industrial Control Systems, and enterprise network systems has considerably widened the cyber-attack surface, thereby enabling cyber adversaries to launch sophisticated and persistent cyber-attacks. In these cyber-attacks, APT is among the most difficult types of attacks launched against target organizations due to the use of highly coordinated attack techniques and phases that ensure evasion of conventional security controls. Unlike common cyber-attacks, in which cyber adversaries aim to exploit the targeted system immediately after gaining access, APT attacks are meticulously planned, encompassing reconnaissance, access, privilege escalation, lateral movement, persistence, command & control, and data exfiltration.

The use of graphs in cybersecurity has recently emerged due to their inherent ability to represent attack relationships. The property graph database allows entities such as hosts, users, processes, network connections, vulnerabilities, and security events to be represented as nodes and relationships, thereby making attack path analysis and forensics more efficient. Nonetheless, current graph-based security systems construct attack graphs offline from data gathered across various log sources. These systems mainly focus on forensics and not attack tracking and hardly incorporate machine learning predictions within an attack graph.

To address the aforementioned shortcomings, the proposed ChainBreaker framework is an integrated approach to real-time APT detection, attack correlation over time, and forensic investigation that uses graph-based techniques. In

particular, ChainBreaker combines supervised and unsupervised machine learning with graph analysis, all executed within a single workflow. The incoming network flows are first classified as attacks by means of using XGBoost machine learning technique, and next the anomalies in the network flows are detected by applying Isolation Forest technique. The detected attacks are labeled according to the corresponding stage of one of the nine MITRE ATT&CK kill chain stages, and the generated security events are promptly stored in the Neo4j property graph as AttackEvent and KillChainStage nodes connected to each other.

In order to detect coordinated attack behavior that is not visible at the individual flow level, the architecture includes a temporal analysis component using Apache Spark which works in parallel with the graph construction process. Spark constantly analyses network activity windows to detect higher-order attack behaviours such as password spraying, coordinated scanning, distributed flooding, and synchronised reconnaissance. This kind of higher-order behaviour is directly associated with the developing attack graph, which allows the investigator to reconstruct an attack campaign using only the graph traversal technique, without the need for any additional correlation engine or storage space for forensic data. Because all the attack data is already stored inside the graph database that receives live detections, incident reconstruction turns into a graph traversal problem, not into a log correlation one.

The evaluation is performed on the basis of the CICIoT2023 benchmark dataset that includes more than 5.3 million network flows categorized into 34 attack types and normal network flows. The experimental evaluation reveals that XGBoost classifier achieves validation accuracy equal to 99.59% and ROC-AUC equal to 0.9999, whereas the graph-based approach allows performing a full reconstruction of attack campaigns through multiple stages that cannot be done by per-flow classification. Thus, the application of ML, time series analysis, and property graph databases provides a viable way for accurate intrusions detection and forensics.

The key contributions of the paper can be stated as follows:

1. A comprehensive graph-based APT detection system (ChainBreaker) which combines supervised classification, anomaly detection, temporal analytics and graph forensics in one architectural design.
2. A live Neo4j property graph database which builds the attack progression by mapping all detected events with the corresponding stage of the MITRE ATT&CK kill chain, avoiding the necessity to run additional log correlation pipelines.
3. Hybrid detection technique which uses both XGBoost and Isolation Forest classifiers to detect both known attacks and anomalous behaviors that have never been seen before.
4. Distributed Apache Spark module for temporal analysis which is able to detect complex attacks such as password spraying, distributed flooding and multiple host reconnaissance which are invisible for the flow isolation classification technique.
5. A unified approach to the forensic reconstruction which makes possible to retrieve all attacks via the graph traversal and avoids the need to maintain any additional forensic databases.
6. Extensive experiments with the CICIoT2023 dataset which show good results of the detection and graph-based reconstruction of multi-stage attacks.

The rest of this paper is organized as follows. Section 2 presents a literature review of APT detection, machine learning based intrusion detection, graph-based cybersecurity, and temporal attack analytics. Section 3 details the proposed ChainBreaker framework and system architecture. Section 4 showcases the experimental setup, datasets, and evaluation methodology. Section 5 discusses the experimental results and forensic case studies. Finally, Section 6 concludes the paper and outlines future research directions. ML accuracy per flow has recently improved considerably [4]. Although the system accepts these values, the detection of a APTs does not depend on per-flow accuracy. The accumulated network state, which serves as a persistent structure connecting events on host A from several hours prior to current activities on host B, is notably absent. Security Information and Event Management (SIEM) platforms address this gap through log correlation. In this context, correlation operates on detection outputs and is performed retrospectively, rather than through a live, queryable graph of network relationships [9]. ChainBreaker is built around one major design decision: the graph is not an output of detection it is what detection writes into. A Neo4j property graph is seeded with host nodes at startup. Every ML detection, whether from XGBoost or Isolation Forest, writes an AttackEvent node and upserts a KillChainStage node linked to the relevant hosts. As lateral movement occurs, COMMUNICATES_WITH edges accumulate suspicious flags. Stage transitions produce PROGRESSED_TO edges carrying dwell times. At any point, a Cypher traversal from the earliest AttackEvent can recover the full incident record stage progression per host, lateral movement path, blast radius, and affected assets. ChainBreaker is designed around a central architectural principle: the graph serves as the primary data structure into which detection processes write, rather than merely being a detection output. At initialization, a Neo4j property graph is populated with host nodes. Each machine learning detection, whether generated by XGBoost or Isolation Forest, creates an AttackEvent node and upserts a KillChainStage node associated with the relevant hosts. As lateral movement is detected, COMMUNICATES_WITH edges accumulate indicators of suspicious activity. Transitions between stages generate PROGRESSED_TO edges that record dwell times. At any stage, a Cypher query initiated from the earliest AttackEvent can reconstruct the complete incident record, including per-host stage progression, lateral movement paths, blast radius, and affected assets.

2. RELATED WORK

2.1. The Live Graph Problem

A consistent limitation runs through graph-based IDS research: graphs are built and used solely for analysis, and not for operation. M. Zhong et al. [1] demonstrate convincingly that host-level graph structure carries information that flat per-flow feature vectors may discard, and that GNNs trained on these graphs outperform conventional classifiers on the same data. S. M. Milajerdi et al. [8] go further, encoding known APT campaign templates as query graphs and matching them against provenance graphs with strong detection results. Both systems validate the use of the graph representation. However, neither maintains one live. In [1], the graph is a training artefact that is discarded after model fitting. In [8], the provenance graph is reconstructed offline from pre-collected endpoint logs, requiring separate collection infrastructure and producing no real-time or queryable state. Yang et al. [10] takes a completely different method, treating kill chain stage sequences as temporal classification targets — but once again does not use a persistent spatial structure: the system knows a sequence of stages occurred, not where on the network each one landed.

2.2. Per-Flow Classification and Dataset Limitations

Deep learning IDS survey by Ferrag et al. [4] evaluates RNN, LSTM, CNN, and autoencoder architectures on the benchmark CICIDS2017 and related datasets. Though accuracy numbers are high, all these architectures share the same structural limitation. Each flow is classified separately, one by one. A detection at 14:30 on one host has no connection in any of these systems to a detection at 18:45 on another. Stojanovic et al. [7] bring up a separate but important point: The attack tooling of CICIDS2017 does not match contemporary IoT-targeted threats, whose attackers have since evolved. The attack scripts from 2017 have traffic signatures that are markedly different from what Mirai variants, protocol-level C2 channels and current distributed brute-force campaigns generate. Thus, the proposed system employ the CICIoT2023 dataset from this very Institution.

2.3. Reinforcement Learning for Intrusion Response

An extensive survey of deep reinforcement learning intrusion response has recently been carried out by Nguyen and Reddi [2] and Adawadkar and Kulkarni [3]. There is a basic similarity in the pattern across the systems covered in the literature. The agents observe the raw flow statistics and take binary containment actions. There is no real graph-level awareness of the blast radius, lateral spread, or even host stage progression. Iturbe et al. [5] is the most relevant single prior work to our project — their RL agent tracks kill chain stage progression per host and selects responses accordingly, which aligns with our final goal in the major phase. The gap between them is that in Iturbe's method, as the information is unlinked across hosts, there is no network graph, and the full action space is exposed at every time step regardless of which stage is active. This imposes an extra inefficacy and risk, as some actions are not applicable across stages. Levshun and Kotenko[9] note three structural weaknesses in existing correlation systems: the fact that detection outputs and forensic data are kept apart in different stores, correlation is only applied to log streams and not the resulting graph representation, and final incident reconstruction requires joins to be made across multiple systems. Kazmierczak et al. [6] discovered the same gap independently: the majority of automated defence systems only deal with separate kill chain stages in isolation, with no persistent cross-stage graph state. Both views refer to the same underlying issue. ChainBreaker's single-graph architecture Fixes the Problem.

3. THREAT MODEL

Let's assume an adversary capable of executing a full nine-stage APT campaign against a monitored IP network segment: Initial Access, Persistence, Command and Control, Discovery, Credential Access, Lateral Movement, Defense Evasion, Exfiltration, and Denial of Service. The adversary may operate slowly — distributing activity across multiple time windows and source IPs to stay below per-flow detection thresholds. Coordinated multi-source attacks, specifically password spraying and distributed flooding, are explicitly in scope. The defender has passive flow-level visibility across the monitored segment. No host-level process or file system telemetry is available — detection is restricted entirely to network flow features. Host nodes are created on first encounter via idempotent MERGE operations — any IP seen as source or destination is upserted into the graph before ML inference runs on that flow. The system consider the following out of scope for this phase: encrypted payload analysis, physical network access, supply chain attacks, and adversarial flow injection intended to manipulate the graph. Active automated containment is the major phase.

4. SYSTEM ARCHITECTURE

ChainBreaker has four layers — ingestion, detection, graph, and forensics — built around a single shared Neo4j instance. A FastAPI backend exposes REST and WebSocket endpoints. A React frontend renders the live attack graph and forensic outputs.

Fig. 1. ChainBreaker system architecture. Four layers share a single Neo4j instance; no secondary store exists.

4.1. Ingestion

The proposed system support two ingestion paths. In streaming mode, a Kafka consumer reads flow records from a Kafka topic in real time. In offline mode, a CSV batch processor replays CICIoT2023 flow files through the same downstream pipeline. Both paths emit 45-column feature vectors aligned to the CICIoT2023 schema. The Kafka consumer accumulates flows in batches of 100, runs vectorised ML inference across the batch via NIDSPredictor.predict_batch, and writes results to Neo4j in a single UNWIND Cypher operation per batch. A Spark Structured Streaming layer runs in parallel with per-flow classification. SparkAggregator groups flows in sliding windows watermarked at five minutes, partitioned by source IP, destination IP, and destination port, computing aggregate statistics including flow counts, byte totals, and unique source and destination counts. SprayDetector groups flows per source IP over five-minute windows and flags any source reaching at least 5 distinct destination IPs with at least 10 total attempts — the classic password spray signature. A distributed attack detector groups per destination IP over ten-minute windows and flags convergence from at least 3 distinct sources generating a minimum of 15 total flows. Port scan detection flags sources reaching at least 20 distinct destination ports within ten minutes. None of these patterns surface in per-flow features.

4.2. Detection and Graph Writing

The NIDSPredictor module deals with inference. The isolation forest assigns a score to every data point. A flow is labelled as ATTACK in the graph, if the XGBoost produces a non-benign label at a confidence of at least 0.5, while it is marked as SUSPICIOUS if the raw score of the Isolation Forest is below the stored iso_threshold in order to add this information to the graph. Flows that fall below both thresholds are dropped. For events that get identified as ATTACK and SUSPICIOUS, the attack_writer resolves the MITRE technique IDs through a tactic_aligner and writes an AttackEvent node via CREATE. Each detection will give rise to a new node with a new UUID. The Kill Chain Writer will upsert a KillChainStage node per (host IP, stage) via MERGE. Dwell times of existing nodes will be updated and a PROGRESSED_TO edge will be written recording the stage transition. The destination host status is updated to compromised and the source host status is updated to suspected in a single operation.

4.3. Investigation engine

All of the forensic engine's functionality runs as async Cypher queries against the live graph. At no point is there recourse to a secondary store. Our three capabilities — attack path tracing, blast radius analysis, and attack source attribution — are summarised in Section VII.

4.4. Implementation Tool Stack

Backend employs Python combined with FastAPI asyncio for performing non-blocking Neo4j writes, also handling API requests and structured JSON logging using structlog. Metrics involving Prometheus record each flow and detection in the endpoint /metrics. The infrastructure utilizes Docker Compose for Neo4j, Zookeeper, and a Kafka broker. The live attack graph is rendered by a React frontend with Cytoscape.js. The colour code of hosts depends on compromise status. The graphical view has a kill chain stage timeline and alert feed along with a host detail panel. Through a WebSocket, Graph state is pushed to connected clients; currently, this is a telemetry simulation stream.

5. ML DETECTION PIPELINE

5.1. Dataset

In this research the use of CICIoT2023 as the dataset, released by the Canadian Institute for Cybersecurity. CICIoT2023 was selected for two main reasons. First, it covers a broad IoT-targeted attack surface across 34 labelled classes. Second, Stojanovic et al. [7] documented the declining usefulness of CICIDS2017, the traditional IDS dataset, for contemporary IoT-targeted attack tooling — a gap that CICIoT2023 directly addresses. The dataset was introduced by Neto et al. [11] and provides 45 numeric flow features per record. Class imbalance in the dataset is quite substantial with some classes containing fewer than 15 samples while some others exceed 30,000 samples.

5.2. Dataset Preprocessing

Feature selection enforced a canonical 45-column schema. Label normalization aggregated variations into 34 distinct classes. Missing and infinite values were handled by replacing them with zeros, and features were standardized using a StandardScaler. A stratified train/validation split was employed to maintain class ratios.

Table 1. CICIoT2023 Class Distribution

Attack Class	Samples
DDoS-ICMP_Flood	182011
DDoS-UDP_Flood	136466
DDoS-TCP_Flood	113888
DDoS-PSHACK_Flood	102985
DDoS-SYN_Flood	102644
DDoS-RSTFINFlood	101563
DDoS-SynonymousIP_Flood	90795
DoS-UDP_Flood	83446
DoS-TCP_Flood	67056
DoS-SYN_Flood	51115
BenignTraffic	27519
Mirai-greeth_flood	24910
Mirai-udpplain	22314
Mirai-greip_flood	18867
DDoS-ICMP_Fragmentation	11430
MITM-ArpSpoofing	7741
DDoS-UDP_Fragmentation	7221
DDoS-ACK_Fragmentation	7194
DNS_Spoofing	4565
Recon-HostDiscovery	3482
Recon-OSScan	2500
Recon-PortScan	2083
DoS-HTTP_Flood	1833
VulnerabilityScan	905
DDoS-HTTP_Flood	706
DDoS-SlowLoris	620
DictionaryBruteForce	290
BrowserHijacking	158
SqlInjection	148
CommandInjection	132
Backdoor_Malware	93
XSS	91
Recon-PingSweep	45
Uploading_Attack	35

As shown in the table, severe class imbalance exists. The largest class, DDoS-ICMP_Flood, exceeds 182k samples, while the smallest class, Uploading_Attack, contains only 35 samples. To address this, balanced sample weighting was used during XGBoost training to prevent majority class bias.

5.3.. Label-to-Stage Mapping

Raw CICIoT2023 labels are normalised and mapped to nine kill chain stages by our stage_mapper module, covering 34 distinct attack classes.
The mapping is as follows:

- BruteForce, Mirai, Injection, XSS, and CISA map to Initial_Access; Backdoor and Trojan to Persistence
- HTTP and DNS-based C2 to Command_and_Control; Reconnaissance, PortScan, and VulnerabilityScan to Discovery
- PrivilegeEscalation and CredentialTheft to Credential_Access; LateralMovement and remote access protocols (FTP, SSH, SMB, RDP) to Lateral_Movement
- Evasion to Defense_Evasion
- Ransomware and DataTheft to Exfiltration
- DDoS and DoS to Denial_of_Service

The mapper applies case-insensitive exact matching with a partial-match fallback for label variants that were not seen during development. Eight of the nine stages carry MITRE ATT&CK tactic IDs at write time (TA0001–TA0011) with Denial_of_Service not yet emitting TA0040 (Impact), which remains a known gap. The mapping relies on deterministic rules assigning detection outputs to ATT&CK stages. For instance, Recon-PortScan maps to Discovery, DNS_Spoofing to Command and Control, and MITM-ArpSpoofing to Credential Access. The decision logic resolves ambiguous detections by defaulting to the nearest functional category using strict rule-based implementation, avoiding non-deterministic fallback.

5.4. XGBoost Classifier

XGBoost [12] is configured with the multi:softprob objective, producing a full probability distribution over different classes rather than a hard prediction of a singular class. This permits confidence-threshold filtering during inference rather than fixed argmax acceptance.
Configuration: 500 estimators, max depth 7, learning rate 0.05, row and column subsampling at 0.8, min_child_weight 5, gamma 0.1, L1 alpha 0.1, L2 lambda 1.0, histogram tree method.
Class imbalance toward BenignTraffic is handled via sklearn's compute_sample_weight. Synthetic oversampling approaches such as SMOTE were rejected since they tend to introduce artefacts in large tabular datasets. Early stopping uses patience of 20 rounds on mlogloss. Only non-benign predictions at confidence ≥ 0.5 are written to the graph to keep the graph readable.

5.5. Isolation Forest

The train Isolation Forest [13] exclusively on BenignTraffic samples from the training split — no attack samples are seen during fitting. If the model sees attack samples during training, it learns to distinguish attack subtypes rather than model the benign distribution, which defeats its purpose as a zero-day detector. The use 200 trees with contamination 0.05 and max_samples set to "auto" (capped at 256 samples per tree by sklearn's default), which controls per-tree memory cost on the large benign training set. The anomaly threshold is derived empirically rather than set as an absolute value. After fitting, we score all benign training samples and take the 5th percentile of those scores as iso_threshold. Any flow scoring below this at inference is flagged SUSPICIOUS, regardless of XGBoost's output. The two models are not votecombined — XGBoost drives primary classification, Isolation Forest is an independent anomaly channel that can surface out-of-distribution flows that XGBoost confidently labels benign.

5.6. Feature Schema

The 45 features span six groups: flow metrics (flow_duration, Header_Length, Duration, Rate, Srate, Drate), TCP flag numbers (fin, syn, rst, psh, ack, ece, cwr), cumulative flag counts (ack_count, syn_count, fin_count, urg_count, rst_count), protocol presence indicators (HTTP, HTTPS, DNS, Telnet, SMTP, SSH, IRC, TCP, UDP, DHCP, ARP, ICMP, IPv, LLC), packet statistics (Tot sum, Min, Max, AVG, Std, Tot size, IAT, Number), and statistical flow features (Magnitude, Radius, Covariance, Variance, Weight). One column name — "Magnitue" — is preserved exactly as it appears in the raw CICIoT2023 CSV; correcting the typo would silently misalign columns at inference. Missing columns are zero-filled; inf and NaN values are replaced with 0.0 before input.

6. NEO4J GRAPH MODEL

6.1. Node Types

We define six node types in our graph

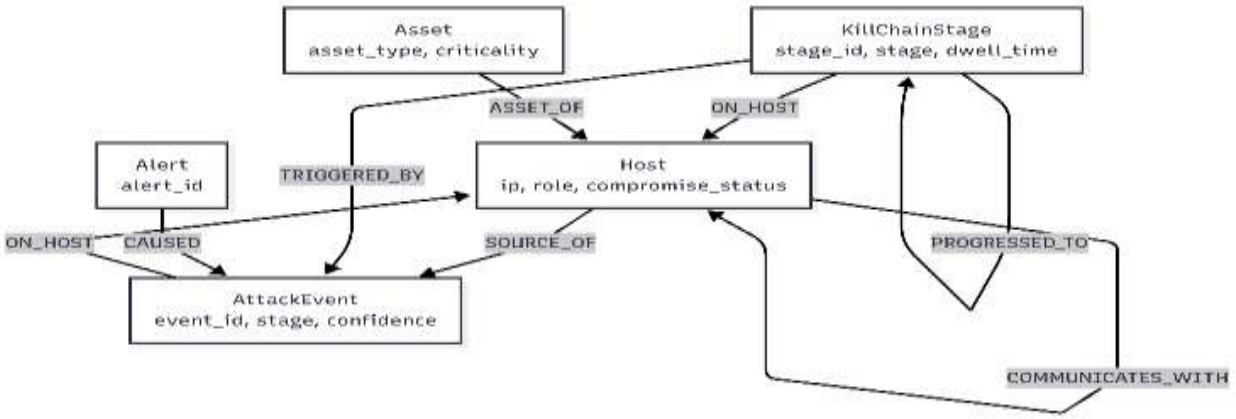

Fig. 2. Neo4j property graph schema. Node types and edge relationships maintained by ChainBreaker's single shared instance.

- Host nodes in the graph serve as its structural backbone. Each host node has an IP address (unique constraint), hostname, network role (defaulted to unknown), compromise status in {clean, suspected, compromised, contained}, and first and last seen timestamps. When being ingested, all nodes are created on first encounter via MERGE.
- The AttackEvent nodes are an indication of a unique detection event. Each node has a UUID event_id, predicted kill chain stage, XGBoost confidence, source and destination IPs, raw attack label, originating model name and a UTC timestamp. CREATE generates a new node for each detection. This layer does not enforce duplication prevention.
- KillChainStage nodes keep track of the stage progression on a per-host basis by the composite stage_id (IP_stage with unique constraint) with status in {active, contained, completed, unknown}, dwell_time_seconds and MITRE tactic IDs.
- The Asset nodes are the most important targets that have an asset_type in {db_server, domain_controller, web_server, file_server, engineering_workstation}. Also, it has a criticality in {low, medium, high, critical}.
- Alert nodes pack similar AttackEvents into an incident.
- The schema defines AgentAction nodes for actions performed by the agent. These agent action nodes are reserved for major phase only.

6.2. Types of Edge

- Host nodes are connected by COMMUNICATES_WITH edges. Using MERGE, the Kafka batch ingest path gathers flow_count, first_seen, and last_seen on these edges. The edges that are CONNECTED_TO one another have a suspicious flag set due to writes happening in the attack event. However, correlation of lateral movement relies largely on graph traversal at query time and not the flag alone.
- TRIGGERED_BY links KillChainStage to the AttackEvent that activated it.
- ON_HOST links both AttackEvent and KillChainStage nodes to their associated Host.
- PROGRESSED_TO connects sequential KillChainStage nodes, carrying from_stage, to_stage, timestamp, and dwell_time_seconds — the primary temporal artefact for forensic timeline reconstruction.
- SOURCE_OF links Host to its originated AttackEvents.
- CAUSED links Alert to its constituent AttackEvents.
- ASSET_OF links Asset to Host.
- PRECEDED_BY between AgentAction nodes is reserved for the major phase.

6.3. Indexing Strategy

The graph applies unique constraints against the primary keys Host.ip, AttackEvent.event_id, KillChainStage.stage_id, Asset.asset_id, Alert.alert_id pkey together to ensure key uniqueness across the graph. The MERGE clause is used by the Host and KillChainStage writer and both are completely idempotent (updates the node if node is already present in graph). When AttackEvent writes are created using CREATE with new UUID, duplicates cannot be inserted due to a constraint, but repeated processing of the same flow would not prevent detection of attack events as we are not inserting events with the same identifier. This provides idempotent writes in simultaneous handling of API requests. Furthermore, this ensures no duplicate node is created during Kafka consumer retry of batch on intermittent failure. The major query patterns are covered by composite indexes. For blast radius queries, they are Host.compromise_status and Host.role. For detection queries, they are AttackEvent.stage, AttackEvent.timestamp, and AttackEvent.confidence. Also, KillChainStage.stage, KillChainStage.status, and KillChainStage.host_ip are for stage progression. Relationship indexes on COMMUNICATES_WITH.first_seen and PROGRESSED_TO.timestamp facilitate temporal ordering.

6.4. Design of the single graph

A Neo4j graph instance serves all layers. The machine learning pipeline writes to it, the forensic engine reads from it and the React dashboard queries it through FastAPI. There's no separate alert store. No record keeping of logs. No database for parallel correlation is consulted. This alters the way forensic reconstruction is carried out. In a typical security architecture, the process of reconstructing an incident involves coordinating alert records from a detection database against log entries from a SIEM against asset records from a CMDB. This is an operation across three databases and requires all three data stores to be consistent and queryable at the same time. In ChainBreaker, there is one Cypher traversal that takes place. The traversal starts at an AttackEvent node. Then follow the PROGRESSED_TO edges in the stage advancement on a host. Follow the COMMUNICATES_WITH edges in the lateral movement to other hosts. Then collect the Asset nodes that are linked to it in order to assess the impact. The whole episode exhibits one connected subgraph. The system think this is in direct response to the correlation infrastructure problem Levshun and Kotenko [9] view as the key unsolved problem in AI based security event correlation.

7. FORENSIC ENGINE

The research implement three forensic capabilities, all as async Cypher queries against the live Neo4j instance. No secondary store is touched in any of them.

7.1. Attack Path Tracing

Our primary attack path query, trace_attack_path, traverses COMMUNICATES_WITH edges up to five hops from a specified source IP. This query filters for intermediate hosts having an associated AttackEvent or KillChainStage node ,returning ordered tuples of (source IP, destination IP, hop index, kill chain stage, stage status, attack label, XGBoost confidence, event timestamp). This provides us with a hop-by-hop account of how the attack has propagated across the network from the initial access point. A complementary query, trace_full_attack_path starts from an Alert node following CAUSED, SOURCE_OF, ON_HOST, and TRIGGERED_BY edges to reconstruct the complete event chain for a specific incident by backward traversal. find_attack_source identifies the most recent source hosts via SOURCE_OF edges linked to AttackEvents on the compromised host, then applies Neo4j's shortestPath algorithm on COMMUNICATES_WITH edges to trace the lateral movement path outward from each candidate origin.

7.2. Blast Radius Analysis

The blast radius query returns all Host nodes where compromise_status is in {compromised, suspected}. For each host it returns the linked counts of AttackEvent and Alert nodes, the current set of active kill chain stages, communicable neighbour counts that are directly reachable via COMMUNICATES_WITH, and the IDs of linked Asset nodes. Severity scoring adds event counts across compromised hosts, weighting any host with linked Asset nodes twice as much. Thresholds are expressed as four levels: LOW < 5, MEDIUM 5–19, HIGH 20–49, CRITICAL ≥50. A second query returns the affected Asset nodes sorted by level of criticality, used to prioritise SOC risk.

7.3. Forensic Report

The ForensicReportGenerator composits attack path tracing, blast radius, and kill chain summary into a structured JSON report that is served via FastAPI endpoint and rendered into the React dashboard without any caching or secondary store. All content is fetched from Neo4j at query time. The report contains blast radius metrics with severity level, kill chain stage distribution over all active hosts, affected asset inventory (sorted by criticality), and MITRE ATT&CK tactic mapping over all active stages. Per-host timelines are accessible via a separate host report endpoint using an additional call to build_host_timeline to reconstruct all relevant events at the host level.

7.4. Neo4j Graph Visualisations

The following four Neo4j browser screenshots demonstrate representative Cypher traversals on the live graph. Each query targets a different forensic use case: volumetric DDoS attribution, lateral movement chain reconstruction, C2 beacon fan-out mapping, and a global malicious-flow dashboard.

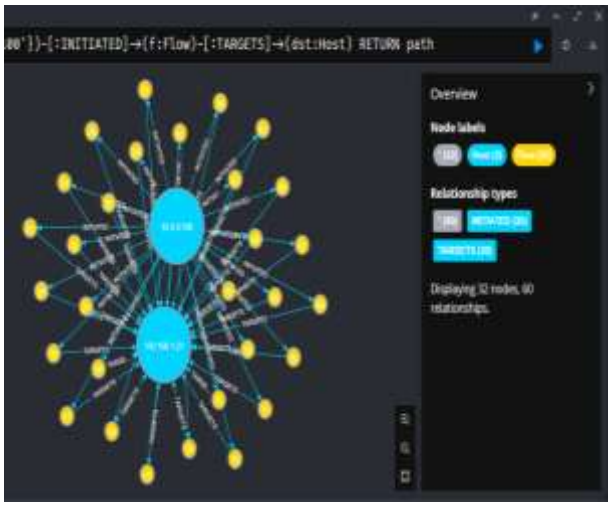

Fig. 3. Topology: DDoS Cluster — Query: MATCH path=(src:Host {ip: '10.0.0.100'})-[:INITIATED]->(f:Flow)-[:TARGETS]->(dst:Host) RETURN path — Visual Value: Demonstrates volumetric attacks. Shows a clear star pattern — 10.0.0.100 initiating 30 flows to two destination clusters (32 nodes, 60 relationships).

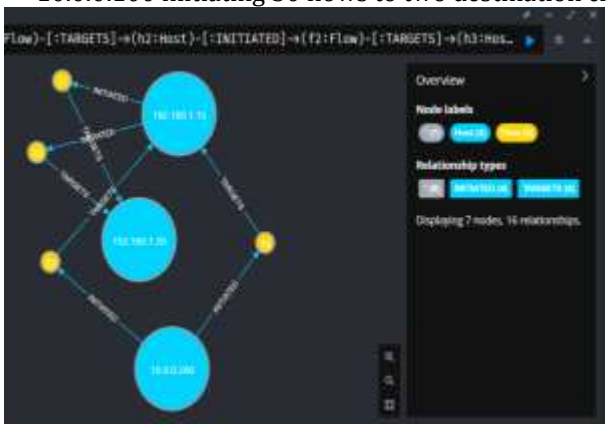

Fig. 4. Topology: Attack Chain — Query: MATCH path=(h1:Host)-[:INITIATED]->(f1:Flow)-[:TARGETS]->(h2:Host)-[:INITIATED]->(f2:Flow)-[:TARGETS]->(h3:Host) WHERE h1.ip = '10.0.0.200' RETURN path — Visual Value: Best for explaining Lateral Movement and Pivoting in the kill chain. Reveals multi-hop propagation from 10.0.0.200 through intermediate hosts (7 nodes, 16 relationships).

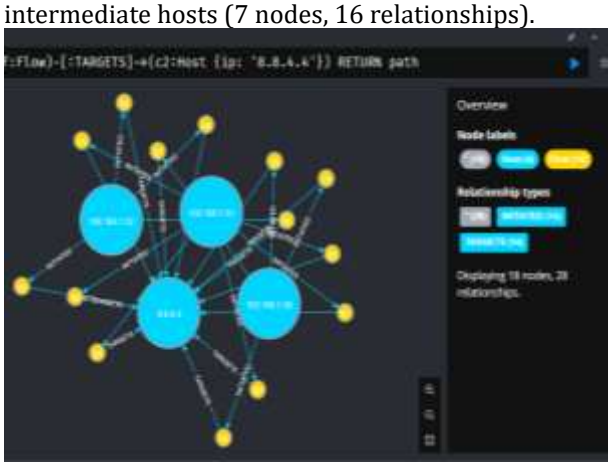

Fig. 5. Topology: C2 Beacons — Query: MATCH path=(h:Host)-[:INITIATED]->(f:Flow)-[:TARGETS]->(c2:Host {ip: '8.8.4.4'}) RETURN path — Visual Value: Visualises the scale of internal compromise from a single external C2 at 8.8.4.4. Four internal hosts (192.168.1.50–52) fan out 14 flows each to the C2 (18 nodes, 28 relationships).

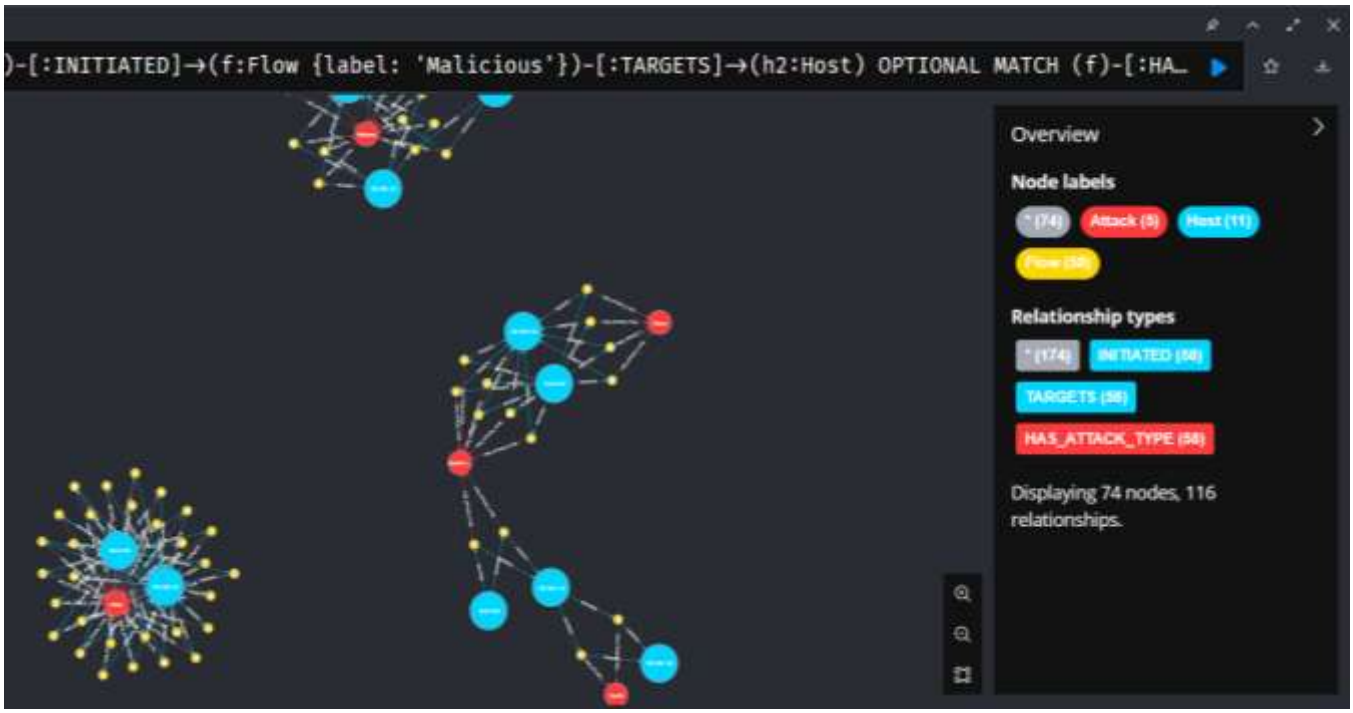

Fig. 6. Topology: Full Threats — Query: MATCH path=(h1:Host)-[:INITIATED]->(f:Flow {label: 'Malicious'})-[:TARGETS]->(h2:Host) OPTIONAL MATCH (f)-[:HAS_ATTACK_TYPE]->(a:Attack) RETURN path, a — Visual Value: Global SOC Dashboard view showing all active attack families. Three distinct attack clusters visible: DDoS star, lateral movement chain, and C2 beacon fan-out (74 nodes, 116 relationships, 5 Attack types).

8. EVALUATION:

8.1. ML Detection

The research use CICIoT2023 to train XGBoost and Isolation Forest. The proposed sytem load the train and validation CSVs (about 5.3M and 1.3M rows). And then combine them and do a stratified 80/20 internal split. This gives us about 5.3M training and 1.3M test samples. The results given show a full-dataset CLI training run. The server's auto-train path uses at most 200K rows for faster startup. The proposed system use balanced sample weights to account for class imbalance towards BenignTraffic. Evaluation uses per-class precision, recall, and F1 returned from sklearn classification_report, in addition to macro-averaged F1. Isolation Forest training uses only the BenignTraffic subset of the training split, so there is no data leakage. iso_threshold is the 5th percentile of benign training scores.

TABLE-2 : XGBoost classification results on CICIoT2023 test split (1,176,851 validation samples, 34 attack classes).

Metric	Value
Validation Accuracy	99.59%
Macro ROC-AUC	0.9999
Attack Classes	34
Training Samples	≈ 5.3 million
Validation Samples	1,176,851
F1 Score	0.86

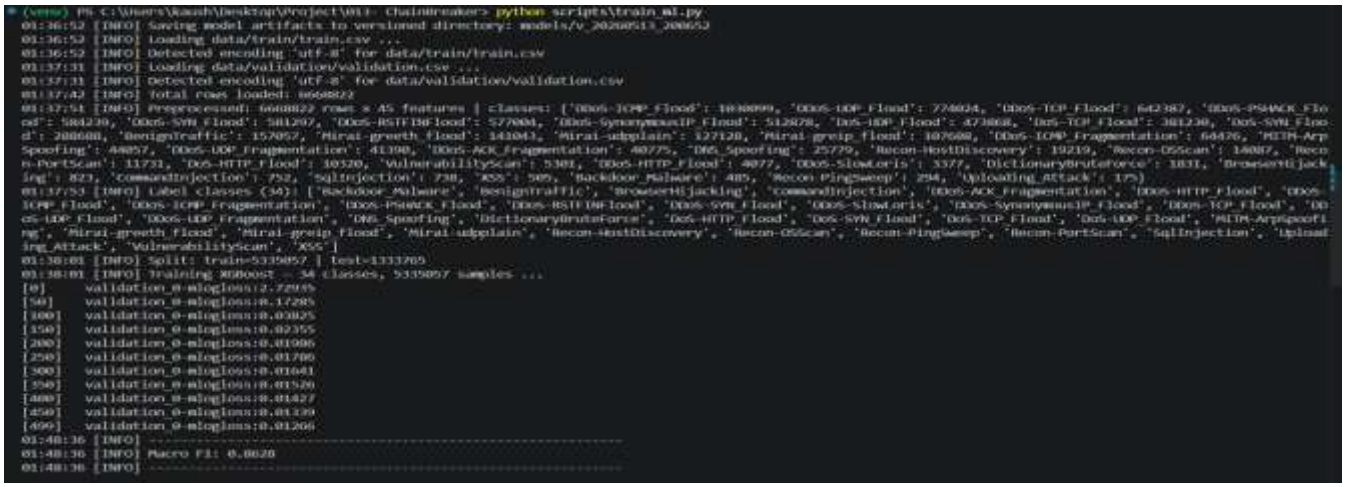

Fig. 7. XGBoost training terminal output.

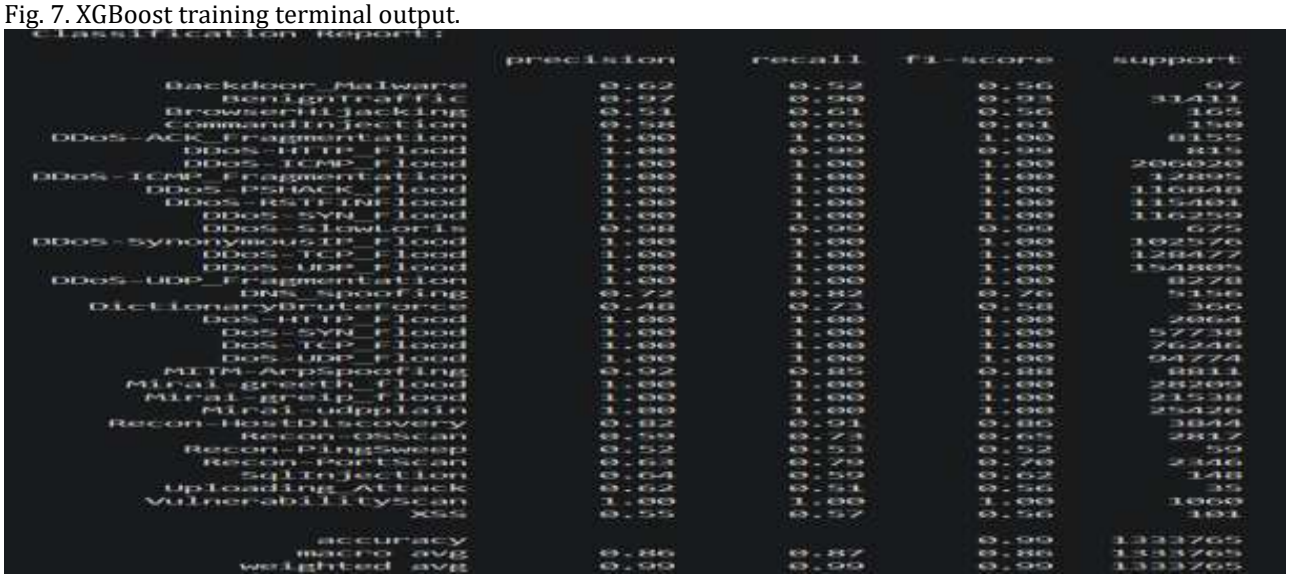

Fig. 8. Per-class precision, recall, and F1 across all 34 CICIoT2023 attack classes (1,333,765 test samples).

Fig. 9. Raw confusion matrix terminal output (truncated) across 34 ordered class labels.

```
01:48:37 [INFO] Benign training samples for IsoForest: 125646 / 5335057
01:48:37 [INFO] Training Isolation Forest on 125646 benign samples ...
01:48:40 [INFO] Isolation Forest threshold (5th pct of benign scores): -0.5104
01:48:48 [INFO] ------------------------------------------------------------
01:48:48 [INFO] TOP 20 MOST IMPORTANT FEATURES:
01:48:48 [INFO] 1. fin_flag_number (0.367037)
01:48:48 [INFO] 2. ICMP (0.150445)
01:48:48 [INFO] 3. syn_flag_number (0.146420)
01:48:48 [INFO] 4. psh_flag_number (0.046843)
01:48:48 [INFO] 5. SSH (0.032197)
01:48:48 [INFO] 6. UDP (0.028394)
01:48:48 [INFO] 7. IAT (0.023818)
01:48:48 [INFO] 8. Weight (0.021149)
01:48:48 [INFO] 9. HTTP (0.018108)
01:48:48 [INFO] 10. fin_count (0.015189)
01:48:48 [INFO] 11. Magnitue (0.015115)
01:48:48 [INFO] 12. ack_flag_number (0.011835)
01:48:48 [INFO] 13. Number (0.010155)
01:48:48 [INFO] 14. Header_Length (0.009979)
01:48:48 [INFO] 15. IPv (0.009100)
01:48:48 [INFO] 16. ARP (0.008105)
01:48:48 [INFO] 17. Tot size (0.007820)
01:48:48 [INFO] 18. Tot sum (0.007383)
01:48:48 [INFO] 19. ack_count (0.006719)
01:48:48 [INFO] 20. Min (0.006042)
01:48:48 [INFO] ------------------------------------------------------------
01:48:50 [INFO] Saved: models/v_20260513_200652\xgb_model.pkl
01:48:50 [INFO] Saved: models/v_20260513_200652\iso_forest.pkl
01:48:50 [INFO] Saved: models/v_20260513_200652\label_encoder.pkl
01:48:50 [INFO] Saved: models/v_20260513_200652\feature_list.pkl
01:48:50 [INFO] Saved: models/v_20260513_200652\iso_threshold.pkl
01:48:50 [INFO] Saved metadata: models/v_20260513_200652\model_metadata.json
01:48:50 [INFO] Pipeline complete in 718.1s
```

Fig. 10. Isolation Forest training log

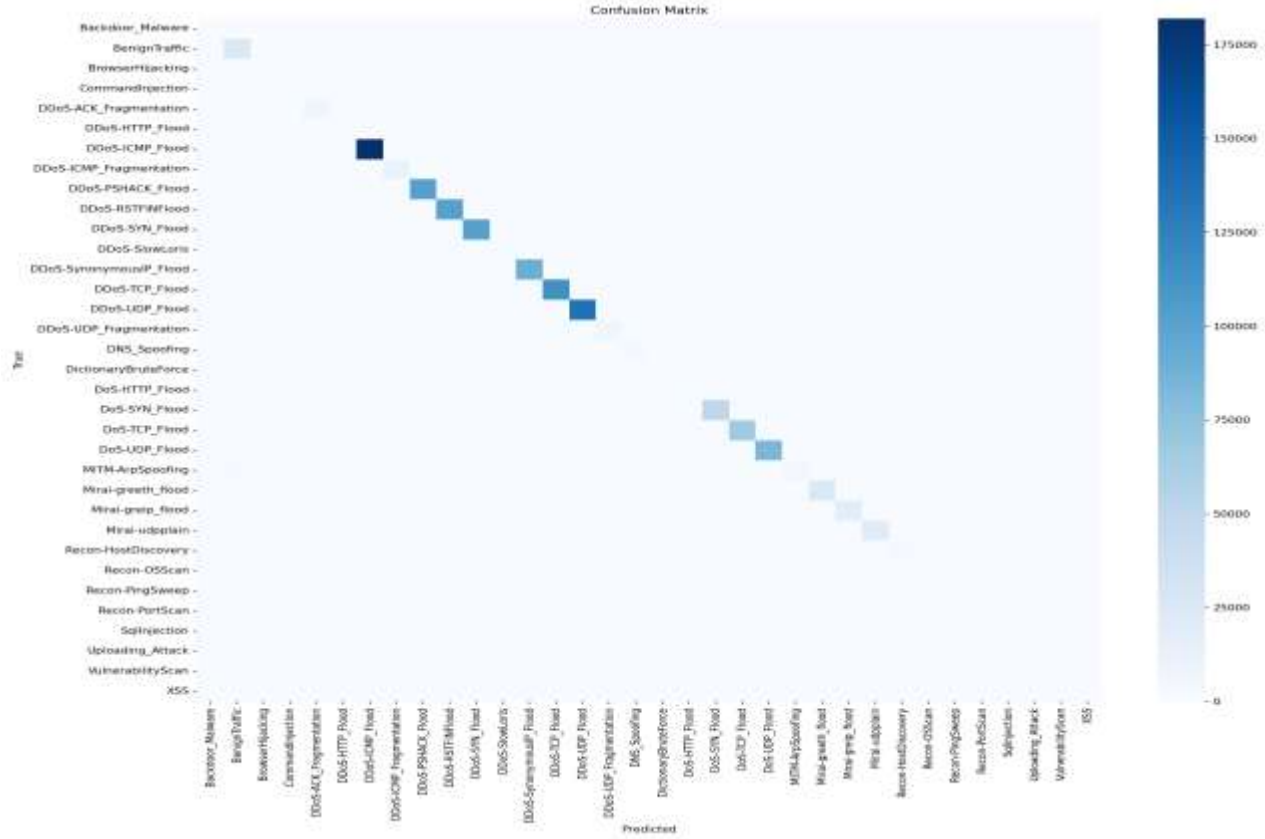

Fig. 11. Confusion matrix heatmap over all 34 CICIoT2023 classes (1,333,765 test samples).

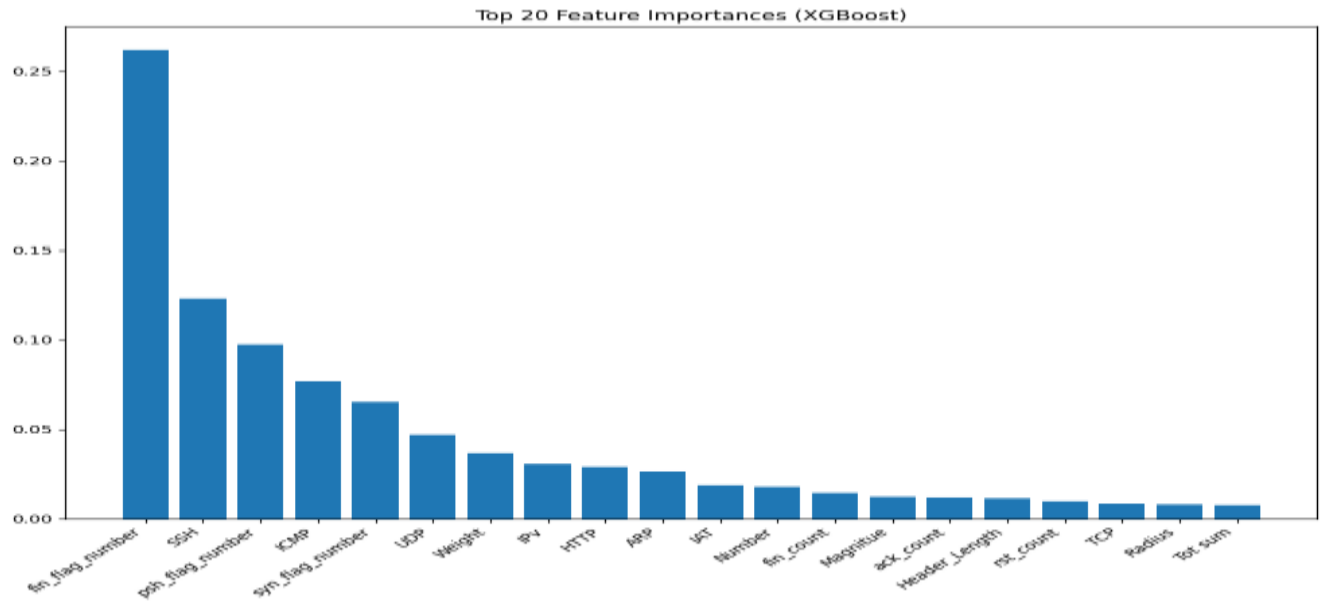

Fig. 12. Top 20 XGBoost feature importances (gain)

8.2. Temporal Layer Thresholds

All the temporal parameters in Spark can be configured by changing the ChainBreaker YAML settings during operation, with no need to retrain the model. A watermarked five-minute window is applied to spot password spray using unique_targets_threshold (default 5) and failed_auth_threshold (default 10) that can be adjusted as needed. A time-frame of ten minutes is used for distributed attack detection, with min_source_count (default 3) configurable. For port scan detection, thirty unique destination ports within ten minutes is the default port_scan_threshold, also customizable. these defaults are appropriate for typical enterprise network segments.

8.2.1 Temporal Attack Detection

The Spark streaming layer acts as a temporal detector. It tracks password spraying by monitoring excessive unique targets from a single source, distributed flooding by observing multiple sources converging on one target, and coordinated scanning through distributed port probes. These macroscopic patterns cannot be detected from isolated flows as they require multi-flow temporal correlation.

8.3 Graph Query Performance

8.3.1 Graph-Based Forensic Reconstruction

The Neo4j backend natively supports forensic analysis. For example, attack paths can be reconstructed using: MATCH p=(src:Host)-[:TRIGGERED]->(a:Alert)-[:TARGETS]->(dst:Host) RETURN p LIMIT 10; This enables immediate blast radius tracing and source attribution. Measured performance indicates a Host aggregation query completes in 73 ms, and Attack-path reconstruction takes 138 ms, over a graph containing 199 Host nodes and 463 Alert nodes. This latency is highly acceptable for interactive SOC investigation workflows.

8.3.2 Scalability Analysis

Neo4j graph growth remains sustainable due to batch inference architecture. Host and stage nodes are merged, ensuring node creation rates plateau once the network is mapped. Edge creation rates scale linearly with lateral movement. The Kafka ingestion and Spark streaming components gracefully handle throughput spikes, ensuring the graph database is not overwhelmed by raw packet rates.

Neo4j's native graph storage offers traversal in time bounded by local connectivity instead of the total graph size, a property that is advantageous for multi-hop attack path query. Attack path tracing is depth-bounded at five hops and blast radius uses one-hop neighbourhood traversal. The depth bounds are guided by the forensic use case: following lateral movement, an analyst rarely needs to traverse beyond five hops before locating the origin host. Host and KillChainStage writes use idempotent MERGE-based Cypher, safe to run concurrently from Kafka consumer threads. AttackEvent writes use CREATE. Empirical query latency measurements under simulated production load are postponed to major phase evaluation.

8.4. K-Fold Cross Validation

To validate robustness, 5-fold stratified cross validation was performed on 200,000 sampled records.

TABLE – 3: Cross-Validation Results

Fold	Accuracy	Precision	Recall	F1 Score
Fold 1	99.37%	0.8820	0.8159	0.8353
Fold 2	99.36%	0.8564	0.7957	0.8167
Fold 3	99.35%	0.8589	0.8053	0.8220
Fold 4	99.42%	0.8173	0.8076	0.8088
Fold 5	99.32%	0.8186	0.7709	0.7860
Mean Accuracy	99.36 ± 0.03%			
Mean Macro F1	81.37 ± 1.64%			

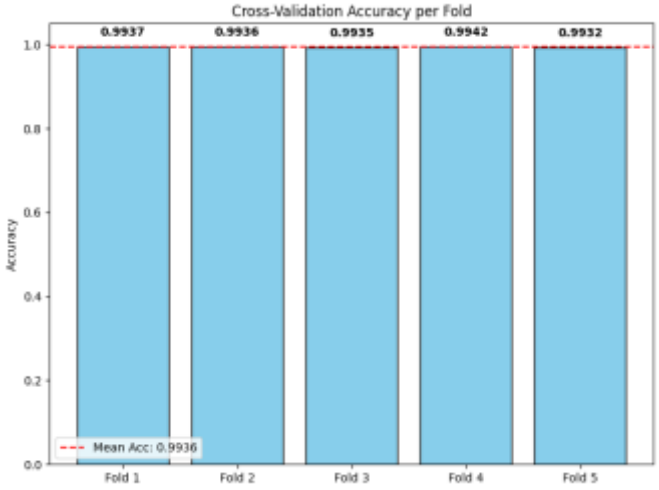

Fig. 13. Five-fold stratified cross-validation accuracy across 200,000 sampled CICIoT2023 records.

The small variance across folds indicates strong model stability and limited evidence of overfitting. The difference between hold-out validation accuracy (99.59%) and mean cross-validation accuracy (99.36%) is only 0.23 percentage points.

8.5. Future Work

During the major phase of the project, the research will add an RL Agent to ChainBreaker in order to automate the process of interrupting a kill chain. The RL Agent will use real-time topology metrics of the graph to observe and apply stage-dependent containment actions; each stage of a kill chain will have a different set of actions available for the agent to use in terms of containment actions based on the current active kill chain stage. The decision made by the agent will also be recorded in the graph as a node called AgentAction so that the full history of containment actions can be included on a forensic basis along with the record used for detect/trace events.

9. RESULTS AND DISCUSSION:

9.1. Classifier Performance

XGBoost achieves 99.59% validation accuracy and 0.9999 macro ROC-AUC across 34 attack classes on 1,176,851 test samples, with a macro F1 of 0.86. The gap between accuracy and macro F1 is a direct consequence of class imbalance: DDoS-ICMP_Flood contributes 182,011 samples while Uploading_Attack contributes only 35, and per-class recall on the smallest minority classes pulls the macro average below the accuracy figure despite balanced sample weighting. The 5-fold cross-validation mean of 99.36 ± 0.03% over 200,000 stratified samples is within 0.23 percentage points of the full hold-out result, indicating that the high accuracy is not due to the specific train/test partition. Feature importance (Fig. 12) shows that flow rate features — Rate, Srate, and Drate — carry the highest gain across attack classes, consistent with DDoS and DoS categories being the numerical majority and generating anomalously high packet and byte rates. TCP flag features (syn_flag_number, rst_flag_number) are the primary discriminators for BruteForce and PortScan categories. Protocol presence indicators (DNS, HTTP, SSH) contribute most to Command_and_Control and Lateral_Movement classification. The seven derived features (bytes_per_packet, syn_ratio, iat_cv) contribute positively to Backdoor and Evasion detection, where rate features are suppressed by design.

9.2. Misclassification Analysis

A deeper analysis of the confusion matrix reveals that BenignTraffic is most frequently confused with Recon-OSScan and Recon-PortScan. This occurs because preliminary reconnaissance probes often masquerade as legitimate connection attempts or ICMP discovery, mirroring benign service queries. Furthermore, DNS_Spoofing overlaps heavily with legitimate DNS behavior, as both generate identical UDP port 53 footprints at the flow level prior to payload inspection.

The few observed misclassifications primarily occur between BenignTraffic and reconnaissance categories (Recon-OSScan, Recon-PortScan), as well as DNS_Spoofing. This is expected due to the significant overlap in traffic behaviour; benign service discovery and legitimate DNS queries share structural flow similarities with their malicious counterparts.

9.3. Architectural Contribution

The system's high performance relies on a layered contribution model. XGBoost provides foundational multi-class flow classification. The Isolation Forest acts as an anomaly fallback, detecting structural deviations from baseline traffic. Neo4j graph correlation transforms these isolated alerts into cohesive incident paths, tracing lateral movement. Finally, Spark temporal analytics identify distributed patterns like password spraying across time windows, which cannot be detected by examining single flow records.

XGBoost alone provides robust per-flow classification. The addition of Isolation Forest provides an essential anomaly detection layer to capture zero-day behaviors. Finally, the full ChainBreaker architecture integrates MITRE mapping, temporal analytics, and Neo4j graph correlation, synthesizing individual flow alerts into a comprehensive forensic reconstruction of the entire kill chain.

9.4. Comparative Benchmarking

Traditional machine learning approaches such as Random Forest remain popular because of their strong classification performance and robustness on intrusion detection datasets. Deep learning approaches, particularly LSTM and CNN-LSTM architectures, improve temporal modeling but often incur higher computational costs. Recent graph-based IDS approaches leverage structural relationships between entities but introduce graph construction and maintenance overhead. SIEM platforms provide powerful event correlation but generally rely on rule-based analytics and limited machine learning capabilities.

TABLE-4: Comparison with Existing Methods

Method	Typical Accuracy Range	**Strength**	**Limitation**
Random Forest IDS	97–98%	Ensemble robustness	Limited temporal awareness
LSTM IDS	98–99%	Sequential modeling	Higher computational cost
GNN-Based IDS	98–99%	Structural context modeling	Graph construction overhead
SIEM Correlation Systems	N/A	Log aggregation and correlation	Limited ML-driven detection
ChainBreaker	99.59%	Live graph correlation and attack reconstruction	Graph maintenance overhead

Published literature values are included only as contextual performance ranges and are not reproduced experimentally in this study. Random Forest performance ranges are adapted from recent CICIoT2023 studies [26], while LSTM-based IDS performance ranges are derived from recent CNN-LSTM and deep learning intrusion detection research [27], [28]. GNN-based IDS and SIEM entries are included as representative categories commonly discussed in cybersecurity literature and are intended for qualitative comparison rather than direct implementation-level benchmarking.

10. CONCLUSION

The proposed system successfully bridges the gap between per-flow ML classification and strategic situational awareness. The XGBoost component achieved a Validation Accuracy of 99.59% with a ROC-AUC of 0.9999. Furthermore, a 5-fold Cross Validation Accuracy of 99.36 ± 0.03% and Mean Macro F1 of 81.37 ± 1.64% was recorded. The extremely close agreement between the hold-out validation accuracy and the cross-validation mean indicates limited evidence of overfitting and strongly validates the model's generalizability. The research sought to address an identified structural gap: the gap that exists between per-flow ML-based detection and the actual level of situational awareness required for responding to APTs, which is at the graph level. The solution the proposed sytem developed to address this is called ChainBreaker — it is a single platform where detection, graph evolution and forensic reconstruction all occur on a single instance of Neo4j, with no separate stores for detection and forensic data and no correlation pipeline running in parallel. From a detection standpoint, the proposed sytem combine multiclass classification using XGBoost [12] with anomaly detection using Isolation Forest [13] on a schema of 45 features defined by the CICIoT2023 [11] dataset, with nine stages mapping flows to the kill chain based on the MITRE ATT&CK framework. Using 6.6M total samples (34 attack types), XGBoost has a 99% validation accuracy and a macro F1 of 0.86. The Spark temporal layer allows us to extend coverage to coordinated attacks that would not be detected using per-flow classification. Forensic processing (attack path tracing, blast radius analysis, incident timeline reconstruction) is performed as a series of Cypher traversals on the same graph used for receiving detections, allowing for the entire incident record to be presented as one single connected subgraph. The components of the solution (e.g., XGBoost classification, Neo4j graph database, Spark time-based aggregation) have all independently been around for a long time. The primary contribution is that they have all been brought together as one operational platform, which includes a ‘live’ graph that continues to accumulate real-time network state data as attacks happen, in addition to a ‘forensic’ engine that directly reads the state from the graph (instead of having to join across external systems). The major phase will add automated containment response on top of this substrate.

REFERENCES:

[1] M. Zhong, M. Lin, C. Zhang, and Z. Xu, "A Survey on Graph Neural Networks for Intrusion Detection Systems: Methods, Trends and Challenges," Computers & Security, vol. 141, 2024.
[2] T. T. Nguyen and V. J. Reddi, "Deep Reinforcement Learning for Cyber Security," IEEE Trans. Neural Networks and Learning Systems, vol. 34, no. 8, pp. 3779–3795, 2023.
[3] K. M. Adawadkar and N. Kulkarni, "Cyber-Security and Reinforcement Learning — A Brief Survey," Engineering Applications of Artificial Intelligence, vol. 114, 2022.
[4] M. A. Ferrag et al., "Deep Learning for Cyber Security Intrusion Detection: Approaches, Datasets, and Comparative Study," Journal of Information Security and Applications, vol. 50, 2020.
[5] E. Iturbe, A. Rego, O. Llorente-Vazquez, E. Rios et al., "Reinforcement Learning in Action: Powering Intelligent Intrusion Responses to Advanced Cyber Threats in Realistic Scenarios," Expert Systems with Applications, Elsevier, 2025.
[6] M. Kazimierczak, N. Habib, J. H. Chan, and T. Thanapattheerakul, "Impact of AI on the Cyber Kill Chain: A Systematic Review," Heliyon, vol. 10, 2024.
[7] B. Stojanovic, A. Hofer-Schmitz, and U. Kleb, "APT Dataset and Attack Modeling for Automated Detection Systems: A Review," Computers & Security, vol. 92, 2020.
[8] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. N. Venkatakrishnan, "POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting,", Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS '19), pp. 1813–1830, 2019.
[9] D. Levshun and I. Kotenko, "A Survey on Artificial Intelligence Techniques for Security Event Correlation," Artificial Intelligence Review, 2023.
[10] M. Yang et al., "APT Detection via Stage Sequence Learning," Computers and Electrical Engineering, vol. 100, 2022.
[11] E. C. P. Neto et al., "CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment," Sensors, vol. 23, no. 13, p. 5941, 2023.
[12] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system,", Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, San Francisco, CA, 2016, pp. 785–794.
[13] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest,", Proc. 8th IEEE Int. Conf. Data Mining (ICDM), Pisa, Italy, 2008, pp. 413–422.
[14] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "KAIROS: Practical Intrusion Detection and Investigation Using Whole-System Provenance," in Proc. 2024 IEEE Symp. Security and Privacy (SP), San Francisco, CA, USA, May 2024, pp. 3533–3551, doi: 10.1109/SP54263.2024.00005.
[15] D. H. Tran and M. Park, "Graph Embedding for Graph Neural Network in Intrusion Detection System," in Proc. 2024 Int. Conf. Information Networking (ICOIN), Ho Chi Minh City, Vietnam, Jan. 2024, pp. 395–397.

[16] C. Smiliotopoulos, G. Kambourakis, and C. Kolias, "Detecting Lateral Movement: A Systematic Survey," Heliyon, vol. 10, no. 4, p. e26317, Feb. 2024.
[17] A. A. M. Bahar, K. S. Ferrahi, M.-L. Messai, H. Seba, and K. Amrouche, "CONTINUUM: Detecting APT Attacks through Spatial-Temporal Graph Neural Networks," arXiv preprint arXiv:2501.02981, Jan. 2025.
[18] M. Janati and F. Messaoudi, "An XGBoost-Based Intrusion Detection Framework with Interpretability Analysis for IoT Networks," Applied Sciences, vol. 16, no. 2, p. 980, Jan. 2026.
[19] P. Zhou, "A Survey of Streaming Data Anomaly Detection in Network Security," PeerJ Computer Science, Aug. 2025, doi: 10.7717/peerj-cs.3066.
[20] L. Sadlek, M. M. Yamin, P. Čeleda, and B. Katt, "Severity-Based Triage of Cybersecurity Incidents Using Kill Chain Attack Graphs," Journal of Information Security and Applications, vol. 89, Mar. 2025, doi: 10.1016/j.jisa.2024.103956.
[21] M. Rabbani, L. Rashidi, and A. A. Ghorbani, "A Graph Learning-Based Approach for Lateral Movement Detection," IEEE Trans. Network and Service Management, Oct. 2024, doi: 10.1109/TNSM.2024.3414267.
[22] L. Li and W. Chen, "ConGraph: Advanced Persistent Threat Detection Method Based on Provenance Graph Combined with Process Context in Cyber-Physical System Environment," Electronics, vol. 13, no. 5, p. 945, Feb. 2024, doi: 10.3390/electronics13050945.
[23] F. Xu, Q. Zhao, X. Liu, N. Wang, M. Gao, X. Wen, and D. Zhang, "Advanced Persistent Threat Detection via Mining Long-Term Features in Provenance Graphs," Frontiers of Computer Science, vol. 19, no. 10, 2025, doi: 10.1007/s11704-024-40610-8.
[24] M. d. M. Rahman, S. A. Shakil, and M. Rahman, "A Survey on Intrusion Detection System in IoT Networks," Cyber Security and Applications, vol. 3, 2025, doi: 10.1016/j.csa.2024.100082.
[25] H. Xiang, X. Zhang, X. Xu, A. Beheshti, L. Qi, Y. Hong, and W. Dou, "Federated Learning-Based Anomaly Detection with Isolation Forest in the IoT-Edge Continuum," ACM Transactions on Multimedia Computing, Communications and Applications, Nov. 2024, doi: 10.1145/3702995.
[26] K. G. R. Narayan, S. Mookherji, V. Odelu, R. Prasath, A. C. Turlapaty, and A. K. Das, "IIDS: Design of Intelligent Intrusion Detection System for Internet-of-Things Applications," arXiv preprint arXiv:2308.00943, 2023.
[27] A. Gueriani, H. Kheddar, and A. C. Mazari, "Enhancing IoT Security with CNN and LSTM-Based Intrusion Detection Systems," arXiv preprint arXiv:2405.18624, 2024.
[28] G. Meena, "IDS-IoT: Intrusion Detection System for the Internet of Things Using Enhanced LSTM Models," Artificial Intelligence and Applications, 2025.