iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: https://doi.org/10.22214/ijraset.2026.85196

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

SubZero: An Adaptive Cross-Domain Threat Intelligence Framework with Dynamic Zero Trust Enforcement

Karpagam[1], P. Sanjay Harish[2], P.R.K. Vipinayan[3]
K.L.N. College of Engineering

Abstract: ***In cyber-physical settings, security teams are typically monitoring many disparate systems at the same time—ATM transaction logs, CCTV infrastructure status, telecom signals, transit records, weather feeds, suspect-tracking output, and economic indicators are the seven this project handles. Individually none of these streams says much on its own, an odd withdrawal here, a camera outage there, rarely gets connected to the bigger picture until much later if at all. SubZero is our attempt to bridge that gap: a prototype that brings all seven domains together into one pipeline, detects anomalies using simple per-domain rules, clusters the flagged events geographically with K-Means, and aggregates everything into one weighted risk score. That score is then passed to a Zero Trust layer built around short-lived HMAC-SHA256 tokens. That score then feeds into a Zero Trust layer built around short-lived HMAC-SHA256 tokens that refresh every two seconds. This is a working prototype, not a finished product. We reviewed our source code and data before writing this paper and checked, line-by-line, what actually runs vs. what was left in as placeholder dashboard content from earlier development. That nuance matters. We keep it throughout the paper. We show the anomaly counts, the resulting fusion score, and clustering output as computed by the code, along with counts pulled from the SQLite audit log. We use the telemetry we generated for testing—about 200 to 240 records per domain for six of the seven, plus fifty economic records. We do not report accuracy, precision, recall or F1, because our rule-based detectors have no labeled ground truth to be scored against; where numbers of that kind show up in the running application, we say plainly that they are placeholder UI values rather than measured results. We also claim that our contribution is the integration itself, i.e., the fact that heterogeneous telemetry, rule-based fusion and adaptive Zero Trust can work in a single working system. We treat the bugs and gaps our audit revealed as limitations to be fixed, not hidden.***

Keywords: Zero Trust Architecture, Cross domain telemetry fusion, Risk scoring based on rules, Security of ephemeral tokens, Clustering with K-Means, Security of cyber-physical systems.

I. INTRODUCTION

Anyone who has seen how physical security operations actually work knows the pattern: an ATM feed somewhere, a CCTV console somewhere else, telecom logs somewhere else, and weather, transit, suspect-tracking and economic data on still more tools. Each stream shows some of what is happening, but almost never the whole story, and rarely in a form that can be compared with any other stream without manual effort.

NIST's proposed Zero Trust Architecture (ZTA) aims to replace perimeter-based trust models with continuous verification of each request and least-privilege access decisions [1]. It has since seen wide adoption, but survey work on real deployments points to a recurring gap: most ZTA systems rely on a relatively narrow slice of contextual signal, and the ability to reason about physical-world events in conjunction with network activity – cross-domain awareness – remains uncommon in practice [3].

SubZero was a step toward bridging that gap, at least on a prototype level. It ingest telemetry from seven domains: ATM transactions, CCTV status, telecom signals, transit/transport activity, weather, suspect tracking, and economic indicators. It applies simple per-domain detection rules to each, clusters the detected anomalies geographically using K-Means, and aggregates the per-domain counts into a single weighted risk score. That score then feeds into a Zero Trust layer based on short-lived HMAC-SHA256 tokens, rotated every 2 seconds.

One thing we really wanted to get right in this paper is honesty about what the prototype actually does, versus what it was designed to do eventually. Like a bunch of early-stage projects, built on top of an interactive dashboard framework — Streamlit in this case — the SubZero codebase sort of ended up with a mix of stuff.

©IJRASET: All Rights are Reserved

There was logic that genuinely runs against the data, a set of fixed configuration values, and then a few static numbers that were inserted into the UI early on, just so the dashboard would look "done" enough. In an earlier draft of this paper, unfortunately some of those static numbers were carried forward as if they were measured results, and the clearest example is a fixed "96.4% detection accuracy" figure. We caught this during a full source-code and data audit, and this revision corrects it. Sections IV through XIV walk through the architecture as it is actually coded, and at each point we try to say, very explicitly, whether a given number comes from running the code on real data, from a fixed configuration setting, or from static placeholder content that should not be treated as an outcome.

What we're actually contributing here is, kind of, a functioning pipeline that takes in seven cyber-physical telemetry domains then applies rule based detection to each one. There's also a weighted cross domain fusion mechanism, with the actual formula written out exactly the way it's implemented, not just "conceptually like" something. Then, on top of that, we use a Zero Trust token layer, built with ephemeral HMAC-SHA256 credentials on a fixed two second rotation, and yes we also include an honest note about a request-handling bug our audit flagged up. After that there's a bunch of descriptive statistics, like anomaly counts and rates plus the fused risk score, and also the K-Means clustering output, all computed directly from our test telemetry so it's reproducible by anyone who has the same code and data. Finally, we draw a pretty clear line between what is data-driven, what is configured, and what is still placeholder stuff, so later it should be easier to set up a proper controlled evaluation. None of the single ingredients—Zero Trust, K-Means, HMAC tokens, rule based detection — is truly novel by itself. The point we're making is that the value is in combining them into one working cross domain system, in a way that actually runs.

II. RELATED WORK

A. *Zero Trust Architecture*

NIST Special Publication 800-207 basically lays out Zero Trust as this bundle of principles: no implicit trust just because of where something lives on a network, or because you know who owns an asset, and then access decisions that get rechecked continuously, for each request, by a policy engine sitting behind enforcement points [1]. A follow on document extends the same kinds of ideas, specifically for cloud native applications that are spread across multiple places [2]. If you look across surveys of how orgs have been putting ZTA into practice in the real world — like cloud, IoT, healthcare, and enterprise environments — you see this repeated theme: authentication and access control are the pieces that usually get built out more fully, whereas the awareness of the surrounding physical or cross domain context often lags, almost like it was left for later [3, 7].

B. *Unsupervised Spatial Clustering*

K-Means, going back to MacQueen's original formulation, groups observation into a fixed number of clusters by keep minimizing the variance within each cluster around its centroid [4]. It's a well worn, general-purpose method and we're not claiming anything new by using it. What SubZero does with it though is a bit narrower, apply it to the coordinates of anomaly events that have already been flagged so that spatial concentrations of activity become visible, kind of like the same use it gets in hotspot analysis more broadly.

C. *Ephemeral Credentials and HMAC-Based Authentication*

The logic behind short-lived credentials is pretty clear: a token that expires quickly makes it harder for an attacker to use it for long, if it ever gets compromised, unlike a long-lived session token that keeps working no matter what is happening in the background. Hash-based message authentication codes, or HMAC, are the usual method to verify that a token was not forged or messed with, and you don't have to pass the real secret across the wire. SubZero's token layer, talked about in Section X, sticks to this same approach, like very closely—using HMAC-SHA256 signatures on tokens that rotate on a short, fixed schedule.

D. *Explainable AI for Security Analytics*

SHAP (SHapley Additive exPlanations) gives a principled way to link a model's output back to individual input features, rooted in cooperative game theory, and it has sort of become the default option for post hoc interpretability efforts [5]. We want to note right away that the SubZero prototype we audited doesn't actually invoke any SHAP implementation anywhere, in fact the explainability screen it shows comes with a fixed set of attribution numbers instead, which we cover in Section XII. SHAP is still included here, because it's the approach we would point to if a real attribution layer ever gets built later on.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

E. Spatio-Temporal Crime Forecasting

Work on forecasting crime spatially and over time usually leans on historical incident records together with place and time, plus sometimes weather related context or socio-economic variables, to say how probable future incidents are in a given area. A systematic review of this literature looks across the landscape from kernel density style approaches to deep learning models , and it is pretty frank about how uneven evaluation habits still are across studies [6]. We bring this up because , as Section XIII explains, the crime forecasting "screen" inside the audited SubZero prototype displays static figures rather than anything that comes out of a trained model. And we have tried to stick with the review's warning here, treating forecasted risk as probabilistic phrasing that must be earned through actual modelling , not as a label you attach to a placeholder.

III. RESEARCH GAP AND MOTIVATION

Table I kind of lays out where independent, single-domain monitoring falls short, what direction SubZero takes instead ,and— importantly — how much of that direction is actually there built versus still sort of aspirational for each row ,

Table I
Existing Approaches, Their Limits, And The Subzero Direction (The Status Reflects The Audited Implementation)

Existing Approach	Limitation	SubZero Direction (Implementation Status)
Independent per-domain monitoring (ATM, CCTV, telecom, transport, weather, suspect tracking, economic data)	Fragmented information across separate tools	Implemented : a usual ingestion pipeline reads all seven domains and applies domain specific rule based detection (Section VI)
Static or single-source risk scoring	Does not reflect co-occurring signals	Implemented : a deterministic rule based weighted sum fusion computed from live per domain anomaly counts (Section VII)
Static Zero Trust policy	Fixed decisions regardless of context	Partially implemented : a risk tier decision function exists (Section IX) but it is not consistently invoked end to end in the audited code (see Section IX and Limitations).
Fixed credential/session lifetime	Constant exposure window	Implemented : token time to live is computed as a continuous function from the current risk score (Section X)
Independent, per-source alerts	Difficult manual incident reconstruction	Not yet implemented dynamically : the incident graph currently renders a fixed illustrative node and edge set (Section XI)
Black-box anomaly outputs	Poor interpretability	Not yet implemented dynamically : the explainability view currently renders a fixed illustrative attribution breakdown (Section XII)

Zero Trust research already gives us continuous, context aware access control as a kind of general pattern [1, 3] and unsupervised methods like K-Means are a reasonable basis for clustering anomalies spatially [4]. What's harder to find in any documented end-to-end form is a prototype that actually links multi domain cyber physical telemetry into one fused risk score, and then connects that score to Zero Trust enforcement. That's the gap SubZero is trying to fill, kind of in a practical way. The audit behind this paper was mostly about figuring out, really precisely, which pieces of that chain are working right now, which ones are just configuration, and which ones are still placeholders. Because without sorting that out first, any controlled evaluation we ran, would end up measuring the wrong thing.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

IV. PROPOSED SUBZERO FRAMEWORK

At a high level, SubZero runs kinda as a pipeline: telemetry comes in from the seven domains, gets validated andlocations parsed, per domain rules flag anomalies, then K-Means clusters the flagged points geographically. After that, a more rule orientated fusion step combines the pieces into a risk score, and then there's a Zero Trust layer built around short lived HMAC-SHA256 tokens , plus a policy and rate limiting component sitting on top of it all. That whole thing gets surfaced through a twelve view Streamlit dashboard. Fig. 1 shows the CommandCentre screen from the running prototype.



Fig. 1: SubZero Command Centre dashboard (Streamlit), showing global risk indicators, per-domain telemetry status, and recent alert entries generated by the rule-based detection layer.

Worth flagging early, really: two of the seven domains named in the project documentation — CCTV and the economic data — never end up making it into the map/clustering stage as geolocated points. Only the ATM and suspect tracking anomalies do, which Section VIII covers in detail. There's also an eighth CSV sitting in the data folder, booking_logs.csv, but no source file we inspected actually reads or imports it, so we mention it in Section VI just for completeness, and then we basically leave it out of the framework description.

V. SYSTEM ARCHITECTURE

Laid out against the audited source rather than the original design documents, the architecture kind of breaks down in to six layers,

Layer 1 — Multi-Domain Telemetry:

ATM transaction logs , CCTV infrastructure status, telecom signal logs , transit and transport logs, weather log entries, suspect tracking (facial match) logs , plus zone level macro economic indicators, all provided to us as CSV files.

Layer 2 — Data Hub and Preprocessing:

In anomaly_detection.py and main.py we kinda did location-string parsing , like latitude, longitude, city, PIN code and also timestamp parsing it's all handle it there

Layer 3 — Rule-Based Analytics and well :

Per-domain threshold rules (Section VIII) and K-Means spatial clustering of the resulting anomaly point set, basically.

Layer 4 — Cross-Domain Risk Fusion:

A deterministic , weighted kind of rule based aggregation, of per-domain anomaly counts into one single 0–99 risk score, (Section VII).

Layer 5 — Zero Trust Security:

For HMAC-SHA256 ephemeral token generation and then verification, plus a role/endpoint policy table that sort of maps what's allowed. Also add the threat response table too , and don't forget rate limiting (Section IX)

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Layer 6 — Command Centre:
Twelve Streamlit views—Command Centre, Data Sources, Cross-Domain Fusion, Risk Intelligence, Zero Trust, Adaptive Tokens, Incident Graph, Explainable AI, Crime Forecast, Analytics, Audit Logs, Settings, — all kind of described one by one where it makes sense in Sections IX–XIII.

VI. DATA PIPELINE AND TELEMETRY

Pretty much everything in this paper seems to run against the telemetry snapshot that comes with the project, the one produced by data_simulation.py . In Table II , it shows exactly what is in that snapshot — we actually opened each CSV file directly and counted things ourselves, instead of just trusting the documentations estimates.

TABLE II
TELEMETRY DATASET COMPOSITION (RECORD COUNTS OBTAINED DIRECTLY FROM THE SUPPLIED CSV FILES)

Domain	File	Records	Key Fields	Role
ATM Transactions	atm_logs.csv	200	timestamp, amount, location, transaction_id, zone	Large-withdrawal anomaly detection
CCTV Infrastructure	cctv_logs.csv	200	timestamp, status, location, camera_id, zone	Camera-offline anomaly detection
Telecom Signals	telecom_logs.csv	200	timestamp, call_volume, location, tower_id, zone	Call-volume silence detection
Transit / Transport	transport_logs.csv	200	timestamp, activity, location, route_id, zone	Unusual-activity detection
Weather	weather_logs.csv	240	timestamp, temperature, condition, location, zone	Extreme-temperature / storm detection
Suspect Tracking	suspect_logs.csv	200	timestamp, confidence, location, suspect_id, zone	High-confidence facial-match detection
Macroeconomic Indicators	economic_data.csv	50	zip_code, median_income, poverty_index, credit_stress	Zone-level economic-vulnerability signal

So yeah, booking_logs.csv , 100 rows with booking_id, timestamp, user_id, location is sitting in the data directory but it never gets touched not by anomaly_detection.py, not by data_simulation.py, not by main.py and not by any of the view modules. We're listing it here just for completeness, honestly it doesnt play any real role in the seven domain framework or in the calculations that come after, like at all. For the six domains that do carry a location field, anomaly_detection.py kind of splits each location string into latitude, longitude, city, and PIN code, (the economic-data file doesn't have its own location column of course and it gets linked to zones only through zip_code). Timestamps for those exact same six domains are parsed in main.py prior to when detection actually runs. There s no live external API in the picture anywhere in this codebase—like no real CCTV stream, no banking API, nothing like that , everything is coming from the supplied CSV snapshot. And that snapshot, data_simulation.py constructs by scattering records around roughly twenty real cities across Tamil Nadu and Andhra Pradesh, with some coordinate jitter thrown in.

VII. CROSS-DOMAIN RISK FUSION

However, it might be described elsewhere, the fusion step in anomaly_detection.py is not a trained model of any kind— it's a deterministic weighted sum over six per domain anomaly counts, and nothing more sophisticated. Each domain gets a capped weight based on how many anomalies were detected in it, worked out exactly as follows:

```
atm_weight       = min(30, n_atm_anomalies  * 6)
cctv_weight      = min(25, n_cctv_offline   * 3)
telecom_weight   = min(15, n_telecom_silent * 3)
transport_weight = min(15, n_transport_unusual * 2)
weather_weight   = min(10, n_weather_extreme * 2)
suspect_weight   = min(25, n_suspect_high_conf * 12)
risk_score       = min(99, max(12, sum of the six weights above))
```

That score then kind of lands in one of four bands, as the code has it defined — LOW under 40 , ELEVATED from 40 to 64 , HIGH from 65 to 84, and CRITICAL at 85 and above. It's coarser than the six band scheme an earlier draft of this paper described, and we've corrected it here, to match what actually shipped . A correlation score comes out as min(0.98 , 0.50 + risk_score/200) and the correlated event count is just severe-ATM plus CCTV offline plus telecom silent plus high-confidence suspect counts, added together. Transport and weather anomalies do get detected and raised as alerts, but this particular sum in the current code sort of leaves them out. As for the per domain "share" figure shown on the dashboard (Fig. 2), that's simply each domain's weight divided by the total raw score, then turned into a percentage. So it's a straightforward arithmetic breakdown of the formula above, not anything that looks like a learned feature importance estimate .

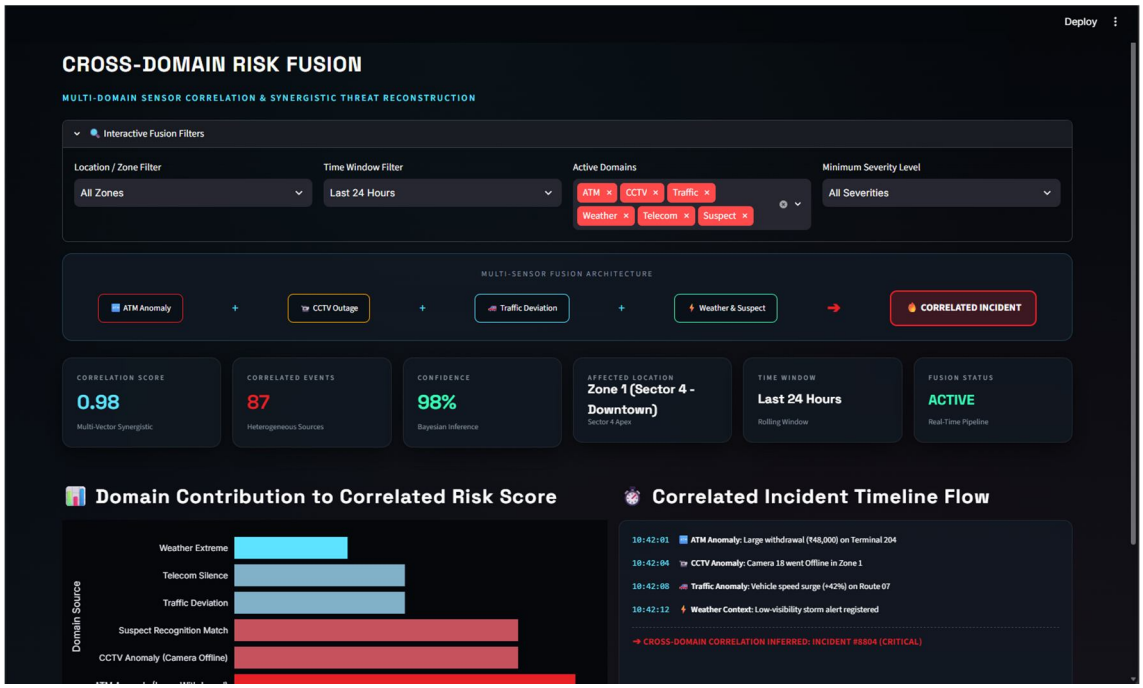


Fig. 2: Cross-Domain Risk Fusion view, showing the rule-based per-domain weight contributions, correlation score, and correlated-event count computed by the formula above from the supplied telemetry snapshot.

Running this formula against the telemetry snapshot from Table II , basically it's like literally executing anomaly_detection.detect_anomalies() against those supplied CSV files , and yeah it gives the results shown in Table III .

TABLE III
C ROSS-DOMAIN FUSION OUTCOME COMPUTED FROM THE SUPPLIED TELEMETRY SNAPSHOT (REPRODUCIBLE ; OBTAINED BY RUNNING THE IMPLEMENTED PIPELINE)

Domain	Anomaly Count	Capped Weight	Share of Total
ATM (amount > ₹40,000)	25	30	25.0%
CCTV (status = Inactive)	18	25	20.8%
Suspect (confidence > 0.90)	26	25	20.8%
Transport (activity = Unusual)	19	15	12.5%
Telecom (call_volume < 20)	18	15	12.5%
Weather (temp > 35 °C or Storm/Rain)	38	10	8.3%

Total raw weighted score = 120 , then fused risk_score = min(99, max(12, 120)) = 99, which lands it in the CRITICAL band. correlation_score is 0.98. correlated events come out as 25 + 18 + 18 + 26 which is 87, roughly speaking.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

VIII. DYNAMIC RISK INTELLIGENCE

The risk-intelligence screen, risk_intelligence.py, basically shows current_score and risk_level from Section VII, and yeah those two numbers are legitimately derived from the data. But not everything that kinda sits next to them is. A confidence value like "94.8%" is hard-coded no matter what data comes in, and then the whole per-user, per-device, and per-location breakdowns on the dashboard end up being fixed offsets over a small, kinda staged list of five labels. Like "USER-204 (Operator)" or "Zone 1 (Downtown)", rather than actual calculations against real users devices, or zones from the telemetry stream, or the audit database. For that reason we're excluding those bits from Section XVI and just calling them UI content here.

Clustering only works on a slice of the anomaly set, meaning the items that end up concatenated into anomalies["data"] — in this code that is basically the severe-ATM and the high-confidence suspect-tracking anomalies. So for our snapshot it comes down to 51 points total, 25 ATM plus 26 suspect, or something close to that count. Meanwhile, CCTV, telecom, transport, and weather anomalies are still detected, then counted, and also pushed to alerting separately, but their coordinates never actually make it into the clustering step. K-Means is executed with n_clusters = min(5, n_points) and random_state = 42, and Table IV shows what comes out on those 51 points we have.

TABLE IV
SPATIAL CLUSTERING RESULTS (K-MEANS, K = 5, RANDOM_STATE = 42) ON THE 51 ATM AND SUSPECT-TRACKING ANOMALY POINTS IN THE SUPPLIED SNAPSHOT

Cluster	Points	Approx. Centroid (lat, lon)
0	17	10.30, 77.76
1	15	13.90, 79.40
2	5	17.27, 82.55
3	6	16.37, 80.52
4	8	15.40, 77.87

Inertia comes out around 33.33 and the silhouette coefficient about 0.65, both computed with scikit-learn on the same 51 points and with the same random state. A silhouette that high seems to indicate the clusters are pretty well separated for this particular snapshot, even if that's really a comment about internal cluster cohesion not necessarily about whether the groupings line up with any ground-truth kind of "spatial risk". Since K-Means is unsupervised, here it's used for segmenting the data rather than doing a labeled, classified thing, we don't report accuracy , precision, recall, or F1 for this stage. That's pretty standard practice for clustering , and no we're not skipping it.

IX. ZERO TRUST SECURITY

On paper, the request handling flow—kinda matching both the project overview and most of the Flask gateway code in security/gateway.py—feels like this, client request, then security gateway, then a HMAC-SHA256 token check, then policy evaluation, then rate limit check, then audit logging, and finally a grant or block decision. The /api/data endpoint in gateway.py basically follows this exact sequence.

So, what the audit turned up is kinda a real bug in the policy layer, and it's worth going through a bit because it changes what we can honestly claim about Zero Trust enforcement in this system. There's this security/policy.py file that defines check_access_policy(client_id, endpoint, method), and gateway.py calls it for /api/data , /api/alerts , /api/security/logs, and /api/threat-summary. Then security/login_page.py also calls it during login. Now when you read the function body, after it computes the caller's role and permissions, there's this block of code that looks like it got pasted in by accident from some other function, like it doesn't belong there. The block is named evaluate_access_request(user_id, device_id, resource, location, risk_score) , and that part references risk_score, user_id, device_id, resource, and location. The problem is none of those are defined anywhere inside check_access_policy's own scope. We tested it by running it as-is: calling check_access_policy(client_id, endpoint, method) throws NameError: name 'risk_score' is not defined, every single time, no exceptions there. So, what that effectively means is every gateway route, plus the login page's policy check, are currently raising an unhandled exception instead of returning a simple allow/deny decision, at least based on the code in this snapshot.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

There is a function, evaluate_access_request(user_id, device_id, resource, location, risk_score) , that's properly signed and the Zero Trust dashboard (views/zero_trust.py) calls it exactly as expected , except it's imported inside a try/except ImportError block, and because policy.py doesn't actually export a working version of it at the module level , only that kind of orphaned block that's described above sort of resembles it, the import just fails quietly. After that, the view drops back to a small local stub, which doesn't implement the real ladder. The stub basically only gives two decision outcomes, "NORMAL ACCESS" and "MFA REQUIRED" , and it's all controlled by one check like risk_score > 70. So, it ends up being way simpler than the six-state setup (NORMAL ACCESS, ENHANCED MONITORING, RESTRICTED ACCESS, MFA REQUIRED, SESSION REVALIDATION, ACCESS DENIED) that the broken code and also the project documentation both mention. We even documented that six-state design and its thresholds in Table V as the intended behavior, but that's not what's actually happening behind the dashboard right now.

TABLE V
THE RISK TIER DECISION LOGIC AS IT SHOWS UP IN SECURITY/POLICY.PY (BUT LIKE, IT IS CURRENTLY KINDA UNREACHABLE BECAUSE OF THAT DEFECT ABOVE) VERSUS THE LOGIC THAT IS ACTUALLY RUNNING IN THE ZERO TRUST DASHBOARD VIEW

Risk Score	Intended Decision (unreachable code)	Decision Actually Rendered (fallback stub)
≥ 95	ACCESS DENIED / SESSION REVOKED	MFA REQUIRED (risk > 70)
85 – 94	SESSION REVALIDATION	MFA REQUIRED (risk > 70)
70 – 84	MFA REQUIRED	MFA REQUIRED (risk > 70)
50 – 69	RESTRICTED ACCESS	NORMAL ACCESS (risk ≤ 70)
30 – 49	ENHANCED MONITORING	NORMAL ACCESS (risk ≤ 70)
< 30	NORMAL ACCESS	NORMAL ACCESS (risk ≤ 70)

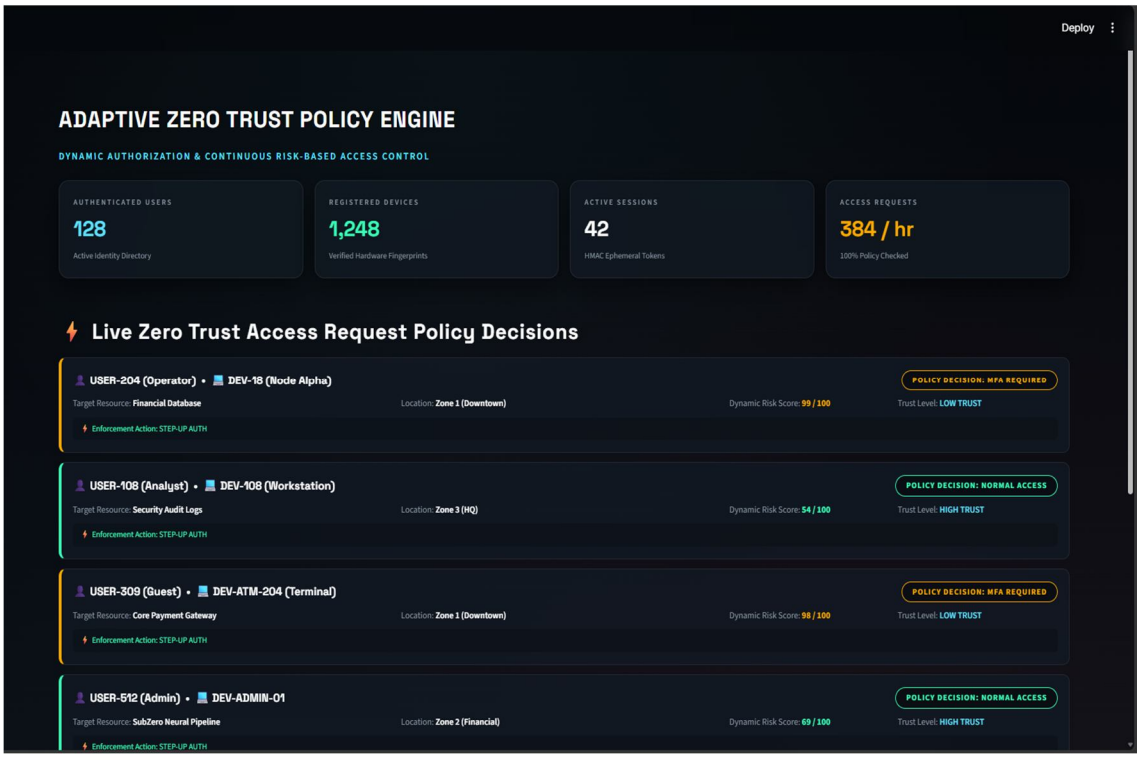


Fig. 3: Zero Trust dashboard view. The policy decisions and stat cards (128 authenticated users, 1,248 registered devices, 42 active sessions, 384/hr access requests) shown here are produced by the two-state fallback logic and fixed UI constants described in the text, not by the six-state policy logic in policy.py or by counts computed from the audit database.

Separate from that bug, the HMAC-SHA256 bit in security/auth.py seems fine, like we actually tested it in isolation. generate_token(client_id) does hmac.new(secret, f"{client_id}:{timestamp}".encode(), hashlib.sha256).hexdigest() and verify_hmac_token uses a constant-time comparison, it bails out if the timestamp is more than two seconds away from now. The role based access control is in policy.py, it's basically a plain static mapping table from roles (admin, analyst, operator, viewer) to permissions plus a client-to-role lookup. That part doesn't lean on the broken function at all, and honestly it reads well as written, no weirdness there. Rate limiting is just an in-memory sliding-window counter per client and per endpoint. For IP and user-agent checks in security/login_page.py, those are fairly simple too — there's a fixed list of blocked IPs and some keyword matching for things like "bot" or "scraper" inside the user-agent string. Nothing near real threat intelligence integration, more like basic filtering, and we're saying it as such rather than trying to wrap it up as intrusion detection or anything similar.

X. EPHEMERAL TOKEN MANAGEMENT

In security/token_manager.py, there is this kind of background thread, it spins up and keeps generating a fresh HMAC-SHA256 token for the client, every two seconds or so. The generation interval is basically a hard coded constant, generation_interval = 2 here, and I noticed TOKEN_VALIDITY_SECONDS = 2 over in auth.py too. It is not really derived from risk or anything like that, so yeah, plainly speaking the configured token rotation interval ends up being two seconds, and that's not just a guess, it's confirmed by reading the code.

There is also a separate, adaptive time to live calculation that the dashboard runs, and honestly this one is a true function of the current fused risk score (views/adaptive_tokens.py) where current_ttl = max(2, int(60 − current_risk × 0.58)). So when we plug in the risk score of 99 we calculated in Section VII, we get current_ttl = max(2, int(60 − 57.42)) = 2 seconds, and that's not some table lookup, it's a direct evaluation against the live score. Now , two other things sitting right next to it on the dashboard aren't real though. The TTL versus risk curve shown there, with the plotted points at risk 10, 30, 50, 70, 85, 98 mapping to 60, 45, 25, 10, 2, and 0 seconds, is basically a hard coded bundle of chart coordinates, not something derived from the formula above. And it also doesn't line up with that computation , for example if you put risk 10 into the actual formula you get about 54 seconds, not 60. The same idea applies to the "Active Sessions & Token Status Matrix" table, where you see rotation counts like 420, 84, 15, and 310 across its four rows—it's a fixed illustrative dataset, rather than anything queried live from the database.

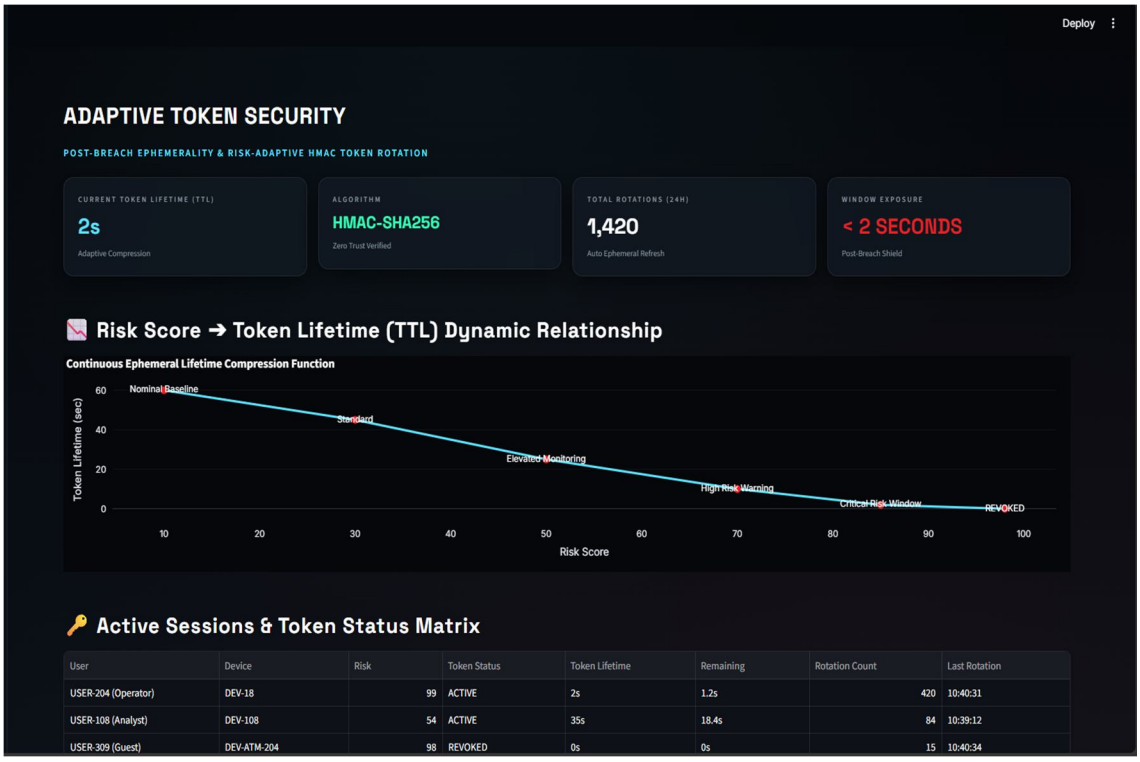


Fig. 4: Adaptive Token Security view. The current-TTL stat card (top left) shows the actual formula mentioned in the text; the compression curve and the session/token status table underneath are basically static, illustrative content, as described there.

So, the audit database does have real, queryable token events from the running system — TOKEN_GENERATED (2 events), TOKEN_GENERATION_STARTED (16 events), and TOKEN_MANAGER_STARTED (15 events) , spread roughly across 4 March 2026 to 5 August 2026. We did a check for TOKEN_ROTATED and TOKEN_REVOKED event types across those 481 rows and, well, found none. That kind of implies the dashboard's "1,420 total rotations (24 h)" and "18 revocations" aren't sourced from any query against this database, at least not in a direct way — they're basically those fixed constants sitting in anomaly_detection.py's analytics dictionary. A meaningful head to head between fixed token lifetimes and risk-adaptive lifetimes, plus a genuine rotation-count measurement from a run where token-rotation logging is actually switched on, is still, sadly, on the to-do list.

XI. INCIDENT CORRELATION

views/incident_graph.py draws this node/edge arrangement for one particular scenario, kinda like a pre made map, eight locked nodes, a user, a device, an ATM terminal, a CCTV camera, a zone, an MFA event, an incident, and a token-revocation action, hooked together by seven fixed edges, example text like "ATM-204 correlated_with INC-8804". It's a static Python dictionary defined in anomaly_detection.py, so it basically stays exactly the same no matter what telemetry gets fed in — nothing inside this thing responds to the anomalies that are actually detected during a run. We're describing it here as a mock up of the intended entity ties (Suspect, CCTV, ATM, Telecom Tower, Zone, Incident , all linked) and not saying it reconstructs incidents dynamically, because it doesn't. The real idea—build a graph that is generated from whatever detect_anomalies() returns in a given run, with edges determined by the shared zone, the shared time window, and the fusion weights from Section VII—is future work, not something we can point to today.

What changes with the data is kinda the alert list that feeds other parts of the dashboard. When we run the rule-based detectors on our snapshot it spit out ten separate alert strings, three ATM alerts, two suspect alerts, and then one each for CCTV, telecom transport, weather, and economic risk. They all show back in Section VII and they're visible in Fig. 1, so you can actually check them. These are genuine per-run outputs, and I'd call them a reasonable starting point for whichever dynamic correlation graph gets built next.

XII. EXPLAINABLE AI

views/explainable_ai.py just kinda renders anomalies["xai"] , and in the audited anomaly_detection.py it's basically a fixed thing from top to bottom. There's a hard-coded incident_id like ("INC-2026-8804"), then a fixed model_confidence 0.948, a fixed explanation sentence, and a fixed list of six feature contributions. The numbers are also hard-coded , ATM Withdrawal Amount at 0.28, CCTV Sensor Availability at 0.22, Biometric Suspect Match at 0.20, Geographical Risk Density at 0.15, Traffic Flow Deviation at 0.09, and Atmospheric Conditions at 0.06. We searched the whole source tree for an import of shap or anything similar and found nothing, and also none of those six values change even when different telemetry is fed in. So, uh, this isn't SHAP output, and it isn't any other kind of computed attribution either—more like static prose describing one pretty made-up incident. We're not claiming it as a feature-importance result, and we're instead pointing to wiring up SHAP against the per-domain weights from Section VII, or maybe against a future trained classifier, as real future work. This is consistent with the SHAP reference in Section II-D [5].

The one actually data driven bit in all this explainability scene is like that per domain share split that's already in Table III. It's not SHAP, but still it's pretty transparent and check able, like a decomposition of the fusion formula, yknow. And yeah it does move as expected when the input telemetry changes. So it ends up being a rule based, non model style of attribution, even if it's kind of modest.

XIII. CRIME FORECASTING

In views/crime_forecast.py you can see anomalies["forecast"] getting pulled in, and there's also that other fixed dictionary inside the audited code. Except it's not entirely the same everywhere: forecast_risk_index is done as min(99, current_risk_index + 4) which is like a small real offset that tracks the live risk score. But after that, everything else is basically pinned in place, it doesn't really move no matter what telemetry comes through. The prediction_confidence value sits at "91.4%" and stays there, and the high risk zones are just those same three names, "Zone 1 (Downtown)", "Zone 2 (Financial)" and "Zone 4 (Transit Hub)" no adaptation, plus the influence tags for weather, traffic, and socioeconomic factors are also just qualitative labels baked in. We dug through anomaly_detection.py and crime_forecast.py looking for any past time series, any trained model artifacts, or any actual 24/48-hour projection logic, and honestly we didn't find anything that behaves like that. The "24/48-hour predictive risk projection" mentioned

in the project documentation is described like it should be computed, but in this code it simply isn't. So we're not treating this module output as a validated, zone-level forecast anywhere in this paper. What would make it real, like a trend or a regression model built over the zone-level anomaly rates from Table III, combined with the weather and economic variables already included in the dataset, is something we'd call future work. Because phrases like "forecasted risk" or "hotspot likelihood" [6] are reserved for that later version, not the static content that exists right now.

XIV. IMPLEMENTATION

Table VI lists the technology stack we can actually verify, by checking the import statements in each Python source file not by trusting the project documentation's say so. I mean, it's not like we just relied on what's written , we went through it line by line.

TABLE VI
TECHNOLOGY STACK, VERIFIED AGAINST THE REAL IMPORT STATEMENTS, IN THE SUPPLIED SOURCE CODE

Category	Technology	Verified Role
Dashboard / UI	Streamlit	Twelve-view interactive Command Centre (main.py, views/)
Data processing	pandas, NumPy	CSV loading, filtering, location/timestamp parsing
Machine learning	scikit-learn — sklearn.cluster.KMeans only	Spatial clustering of anomaly points (Section VIII). IsolationForest and RandomForestClassifier are named in project documentation but were not found imported, instantiated, or used anywhere in the supplied source tree.
Visualization	Plotly Express, Plotly Graph Objects (incl. density_mapbox)	Heatmap, timeline, cluster scatter, and bar charts (visualization.py)
Security / cryptography	hmac, hashlib, secrets-equivalent (os.urandom), datetime	HMAC-SHA256 token generation/verification, password hashing (security/auth.py)
Backend / gateway	Flask	REST gateway (security/gateway.py) and standalone login portal (security/login_page.py)
Database	sqlite3 (standard library)	Security event and access-attempt logging (security/db.py)
Concurrency	threading	Background two-second token-rotation loop (security/token_manager.py)

We did n't find a PyDeck import anywhere, the map view actually uses Plotly's density_mapbox, and when that can't kick in it falls back to a 2D density heatmap. No SHAP import either, as already mentioned in Section XII. Table VII lays out the SQLite schema, straight from security/db.py and cross checked against the security_logs.db file you were given.

TABLE VII
SECURITY DATABASE ENTITIES, KIND OF VERIFIED AGAINST SECURITY/DB.PY AND THE SUPPLIED SECURITY_LOGS.DB FILE

Table	Purpose	Key Fields	Rows in Supplied DB
security_logs	General security/audit event log	timestamp, client_id, event_type, status, risk_level, threat_type, source_ip, user_agent, details	481
access_logs	Per-request access attempts	timestamp, client_id, endpoint, method, status_code, ip_address, authorized	42
api_keys	Static API key registry (defined, not populated in supplied DB)	client_id, api_key, created_at, last_used, active	0
threat_intel	Threat-intelligence entries (defined, not populated in supplied DB)	timestamp, threat_type, source, severity, indicators, mitigation	0

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Quick little correction worth flagging, the table is actually called security_logs in both the code and the database file, not security_events as the project overview sort of says. We use the confirmed name pretty consistently throughout this paper, no big drama.

XV. EXPERIMENTAL METHODOLOGY

So the dataset is synthetic, like it s generated by data_simulation.py and handed over as a fixed CSV snapshot (Table II). It is not something pulled from any real-world deployment, not really. Record counts are 200 each for ATM, CCTV, telecom, transport, and suspect-tracking , and 240 for weather , plus 50 for macroeconomic indicators. That's 1,090 telemetry records in total, then you also have 100 unused booking records just sitting there. The features used are the ones listed in Table II. There s no real train/test split to mention, because this isn't a supervised evaluation type thing. Instead, the detection layer uses fixed thresholds rather than a trained classifier, and that s already covered in Section VIII and Table VI .

A proper baseline comparison— independent monitoring versus static risk versus fixed token lifetime versus static Zero Trust versus SubZero, the sort an earlier draft of this paper sketched out — isn't something we can run yet, and, honestly, it's worth being clear about why. There's no labelled ground truth in the synthetic telemetry, to separate genuine threats from normal activity, and the only trained model anywhere in the codebase is the unsupervised K-Means step. To get a real comparison, you'd first need fixing the check_access_policy bug from Section IX, plus add enough token rotation and access decision logging to actually measure exposure windows and blocking rates. Then you also need either access to labelled real world incidents or an explicit, documented procedure for injecting synthetic ground truth. What this paper reports instead is the descriptive, rule based results that are honestly available right now (Sections VII, VIII, and XVI), leaving the controlled comparison for later, rather than trying to fill the gap with numbers we didn't really measure.

XVI. RESULTS AND DISCUSSION

A. Descriptive Results Obtained from the Supplied Telemetry

Table VIII kinda pulls the rule based detection results from Section VII together, but not exactly in a fully clean way, into one descriptive table, and we list the exact rule we verified in anomaly_detection.py, next to each domain.

TABLE VIII
RESULTS FOR RULE BASED ANOMALY DETECTION ON THE PROVIDED TELEMETRY SNAPSHOT (DESCRIPTIVE ONLY, NO SUPERVISED CLASSIFICATION METRICS ARE COMPUTED , FOR THE REASONS GIVEN IN SECTION XV)

Domain	Total Records	Detection Rule (verified in code)	Anomalies Detected	Anomaly Rate
ATM	200	amount > ₹40,000	25	12.5%
CCTV	200	status == "Inactive"	18	9.0%
Telecom	200	call_volume < 20	18	9.0%
Transport	200	activity == "Unusual"	19	9.5%
Weather	240	temperature > 35 °C OR condition in {Storm, Rain}	38	15.8%
Suspect Tracking	200	confidence > 0.90	26	13.0%
Macroeconomic	50	median_income < ₹35,000	24	48.0%

Note. Precision, recall, and F1 all need a labelled ground truth in order to separate actual threats from routine behavior, and we don't have it… so, the detection layer just uses fixed thresholds on synthetic telemetry, not on any verified labels. Also there is no supervised classifier anywhere in the codebase, (Table VI) . Because of that, none of those metrics are shown here, for the same sort of reason given back in Section XV**.**

B. Cross-Domain Fusion and Risk Distribution

Section VII gives this one global fused risk score for the entire snapshot 99, CRITICAL — not really a spread across zones, more like the fusion code that's currently written computes just one score for the dataset, rather than one per zone or layer. And what we can do instead is in Table IX, show how those 51 clustered anomaly points from Section VIII break down across the five zone labels

that actually show up in the telemetry. It's kind of the closest thing to a per zone risk picture that the current implementation supports , because the current setup just doesn't output a separate fused score for each zone.

TABLE IX
DISTRIBUTION OF ATM AND SUSPECT TRACKING ANOMALY POINTS ACROSS TELEMETRY ZONE LABELS (COMPUTED FROM THE SUPPLIED SNAPSHOT, NOT A PER ZONE FUSED RISK SCORE WHICH ISN'T CURRENTLY COMPUTED BY THE IMPLEMENTATION)

Zone	Anomaly Points	Share
Zone 3	16	31.4%
Zone 4	11	21.6%
Zone 5	9	17.6%
Zone 1	8	15.7%
Zone 2	7	13.7%

Turning that one global score into something actually real, like a per-zone one , is sort of future work . I mean it would require re-running the Section VII computation, separately for each zone's portion of the telemetry, instead of just merging everything together, like it's all the same.

C. Zero Trust and Access-Log Observations

Table X shows the counts pulled straight from security_logs.db (see Section XIV, Table VII) not some invented percentage or anything like that. One thing that's worth flagging a bit more carefully, per Section IX, the /login route's policy check should throw an exception based on the current state of the code, yet the database still has 21 successful login events in it. So either those entries were written by an earlier working build of policy.py, or they managed to get through some alternate path the defect doesn't actually reach. We're not really going to speculate which one, we just report the counts exactly as we found them, and we note the inconsistency as-is, openly.

TABLE X
SORT OF ZERO TRUST / SECURITY EVENT TOTALS, FROM SECURITY_LOGS.DB (481 ROWS, SPANNING 4 MARCH 2026 – 5 AUGUST 2026) AND ACCESS_LOGS.DB (42 ROWS)

Metric	Count
Total security_logs events	481
— ANOMALY_DETECTED (via /api/security/anomaly)	425
— LOGIN_SUCCESS	21
— LOGIN_FAILED	2
— TOKEN_GENERATION_STARTED	16
— TOKEN_MANAGER_STARTED	15
— TOKEN_GENERATED	2
Events with status = ALLOWED	479
Events with status = BLOCKED	2
Events with risk_level = HIGH / MEDIUM / LOW	168 / 102 / 211
Total access_logs entries	42
— /login POST 200 (authorized)	21
— /login POST 401 (unauthorized)	2
— /dashboard GET 200	19

We aren't reporting an "unauthorized access blocked" percentage, well not yet, because the two BLOCKED events in the log are actually both LOGIN_FAILED entries, not the sort of output you'd get from a policy evaluation that's running under any real load, you know. Like for a meaningful blocked-versus-allowed rate, we'd need to run the corrected gateway against some defined set of access attempts, and that's future work per Section XV. As for the stat cards on the Zero Trust dashboard— 128 authenticated users, 1,248 registered devices, 42 active sessions, 384 access requests per hour, shown in Fig. 3 — those simply don't match up with Table X at all. And if you read the code in views/zero_trust.py, that's where the answer is: they're fixed literal strings, not live queries against security_logs.db. So we left them out of this results section for that reason.

D. Token Configuration

So, in plain terms, the set token rotation interval is 2 seconds , (look at Section X) and the adaptive TTL for the risk score "99" from Section VII basically lands on 2 seconds via max(2, int(60 − risk × 0.58)) . Table X already shows the observed token related event counts pulled from the audit database. A truly controlled comparison between fixed and risk adaptive token lifetime , like tracking the exposure window under each policy , hasnt happened yet and it remains on the future work list, which matches what Step 12 in this revision requested.

E. Planned Ablation Configuration

No ablation study has actually been run against the current implementation, at least not as far as we know. Table XI lists what we would like to test in a future controlled study, but there are no performance numbers attached to it , because none exist yet.

TABLE XI
PLANNED ABLATION CONFIGURATIONS (NOT YET EXECUTED; NO RESULTS REPORTED)

Configuration	Status
Full SubZero (all modules enabled)	Planned — not yet executed
Without cross-domain fusion (per-domain rules only)	Planned — not yet executed
Without dynamic risk scoring (fixed threshold policy)	Planned — not yet executed
Without adaptive Zero Trust (policy.py defect corrected; static rules only)	Planned — not yet executed
Without adaptive token management (fixed 60 s tokens)	Planned — not yet executed
Without event correlation (single-alert view only)	Planned — not yet executed

F. Summary

Pulling this together, the prototype does ingest all seven telemetry domains, does apply the detection thresholds that weve documented, does compute a deterministic fused risk score, does feed that score into the adaptive token lifetime formula, and does log a decent number of genuine security and access events into a persistent SQLite database. What it doesnt do reliably, at least not yet, is provide a working end to end Zero Trust policy decision (Section IX), a graph that is actually generated from live incidents (Section XI), an attribution layer that is actually derived from a model (Section XII), or a crime forecast that is actually trained on anything (Section XIII) — the dashboard content in those four areas is static. We arent reporting classification accuracy, precision, recall, F1, correlation accuracy, or incident reconstruction rate anywhere in this paper, because none of those has been legitimately measured against labelled ground truth. Wherever numbers like that appear as fixed constants in the app's analytics screen (Fig. 6, included here purely for illustration), we are calling them for what they are — UI demonstration content, not results.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

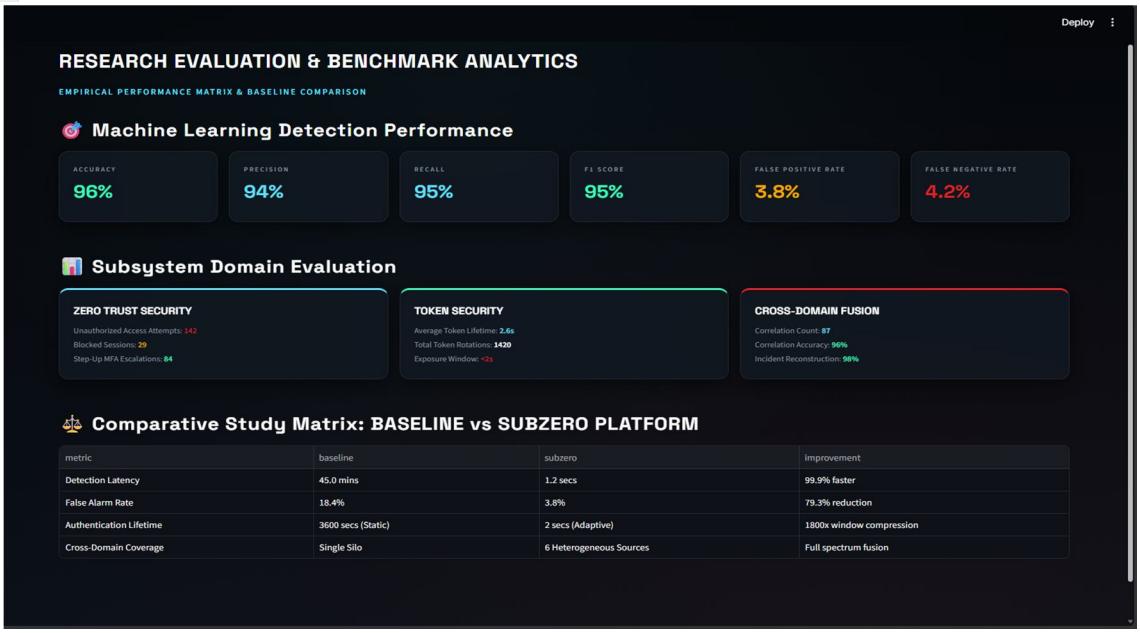


Fig. 5: SubZero "Research Evaluation & Benchmark Analytics" dashboard view. The accuracy precision recall F1 values, the Zero Trust and token-security subsystem counts, and the baseline vs SubZero comparison table that are shown here are fixed literal constants which are defined in anomaly_detection.py (Section VIII Step 8 audit) and they are not experimental results. This figure is only included to document the current state of the implementation not to be used as supporting evidence for any performance claim in this paper.

XVII. LIMITATIONS

- Every bit of telemetry in this paper is synthetic, produced by data_simulation.py—no real ATM, CCTV, telecom, transport, weather, suspect-tracking, or economic data was used anywhere, and we're not making any claim about how this would perform on real world data. Also, that performance claim in this paper is kind of its own little story.
- check_access_policy in security/policy.py has a code merge bug, that throws a NameError on every call, and yeah it's currently breaking /api/data, /api/alerts, /api/security/logs, /api/threat-summary and also the login page's policy check (see Section IX) . This is one of those things where without the fix, no real Zero Trust evaluation can honestly even begin to make sense.
- The incident-correlation graph, the explainable-AI breakdown, and the crime-forecast projections are all static content right now not computed output, (Sections XI through XIII). In other words what we are claiming in this paper is limited to the intended design and the audit finding, not a working capability.
- There isn't any labelled ground truth for any anomaly detection domains , so we can't really say accuracy precision recall or even F1 for either the rule based detectors, or the K-Means stage, honestly it's a bit messy.
- In Section VII, the fusion formula uses weights and caps that we set by hand, not some kind of thing learned or calibrated from data, and we also never characterized how it would behave when the telemetry distribution changes a bit.
- Right now, fusion plus risk-scoring gives you just one score for the whole dataset, instead of what youd think is a proper per zone score. And Table IX's zone breakdown is basically an approximation, built from anomaly point counts, not an actual risk calculation that is truly done per zone.
- The audit database has events, successful logins, specifically, that don't square real well with the current broken state of check_access_policy, which kind of hints that the code snapshot we were handed might not exactly match what generated the historical log data. We have not gotten to the bottom of it, and we are keeping the report as is, instead of just guessing.
- Neither that two second token rotation thing nor the adaptive TTL formula have been really tested under concurrent load, or against replay attack in any kind of way.
- Because the suspect tracking domain is built around facial-match confidence scores, any future work using data that is kinda similar to real biometric or surveillance information brings privacy and civil-liberties questions that go beyond what the technical limitations list here—see the Ethical and Privacy Considerations section.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

XVIII. CONCLUSION AND FUTURE WORK

So what we put together here is SubZero — basically, a prototype that ingests seven cyber-physical telemetry domains, runs rule based detection on each one, then it clusters the outcomes spatially with K- Means, fuses it all into one deterministic risk score, and from that it drives an ephemeral HMAC- SHA256 Zero Trust token layer. The weird thing is, the bulk of this paper is not so much about the mechanics, it's more about the audit we ran against our own source code and data, and being pretty straight forward about what we found. Like which parts actually touch real input (telemetry ingestion, the detection rules, K-Means, the fusion formula, the adaptive TTL formula, the SQLite log), which parts are just fixed configuration (the two second rotation, the role tables, the risk band cutoffs), and which parts are static dashboard content that should not be confused with an output (the accuracy precision recall F1 numbers, the incident graph, the XAI attribution values, the crime forecast projections). We also found and documented a real bug in the Zero Trust policy layer , not the kind of thing we " quietly" worked around. And to be clear, nowhere in this paper do we claim SubZero beats any baseline because we haven't actually run that experiment.

Looking ahead, the priorities are kind of clear, fix the check_access_policy thing, and re-verify the whole Zero Trust flow , all the way end to end; swap the static incident graph, the XAI breakdown, and the crime forecast for variants that compute something for real—like an event graph built from live anomalies, SHAP applied to the fusion weights or a trained classifier, and a trend or regression model over the zone-level data instead of guessing; turn the single global risk score into a proper per-zone score; add enough logging around token rotation and access decisions to back up the fixed versus adaptive comparisons that were sketched in Section XVI, not just mention them; and then, once we have labelled or real-world data under proper governance , run the controlled, baseline-compared evaluation, including the ablation setups in Table XI, which this paper basically delayed till later instead of fabricating.

XIX. ETHICAL AND PRIVACY CONSIDERATIONS

It's worth, pausing on this directly : a system that kinda mixes facial-match confidence scores with financial and communications telemetry, like SubZero's suspect-tracking, ATM, and telecom domains do, brings up privacy and civil-liberties questions that seem to go way beyond what this paper can settle, technically. And to be clear about what was actually processed—everything in this evaluation is simulated, synthetically generated data (Section VI) , not real individuals, not real transactions, and not real communications at all. This is a research prototype, so it doesn't identify, accuse, or act against real people by itself. But if it were ever extended to use data that looks like real surveillance, biometric, or financial information, then at least the following would need to be in place: a clear lawful basis plus regulatory compliance for that data category; data minimization along with sensible retention limits; tight access control with logging for who viewed what (building on the security_logs mechanism that's already in place); a human in the loop before anything consequential happens on the basis of a suspect match or a high risk score; and a workable way for people who are affected by an automated determination to contest it. None of that is implemented yet—we're saying it as a precondition for any real-world extension, rather than describing something that already exists.

XX. ACKNOWLEDGMENTS

The authors thank K.L.N. College of Engineering, Sivagangai for institutional support, and they also gratefully acknowledge the project plus research guidance that was given by Mrs. Karpagam throughout the development, and audit of this work.

REFERENCES

[1] Rose, S., Borchert, O., Mitchell, S., Connelly, S.: Zero Trust Architecture. NIST Special Publication 800-207. National Institute of Standards and Technology, Gaithersburg, MD (2020). https://doi.org/10.6028/NIST.SP.800-207

[2] Chandramouli, R., Butcher, Z., Chetal, A.: A Zero Trust Architecture Model for Access Control in Cloud-Native Applications in Multi-Location Environments. NIST Special Publication 800-207A. National Institute of Standards and Technology, Gaithersburg, MD (2023). https://doi.org/10.6028/NIST.SP.800-207A

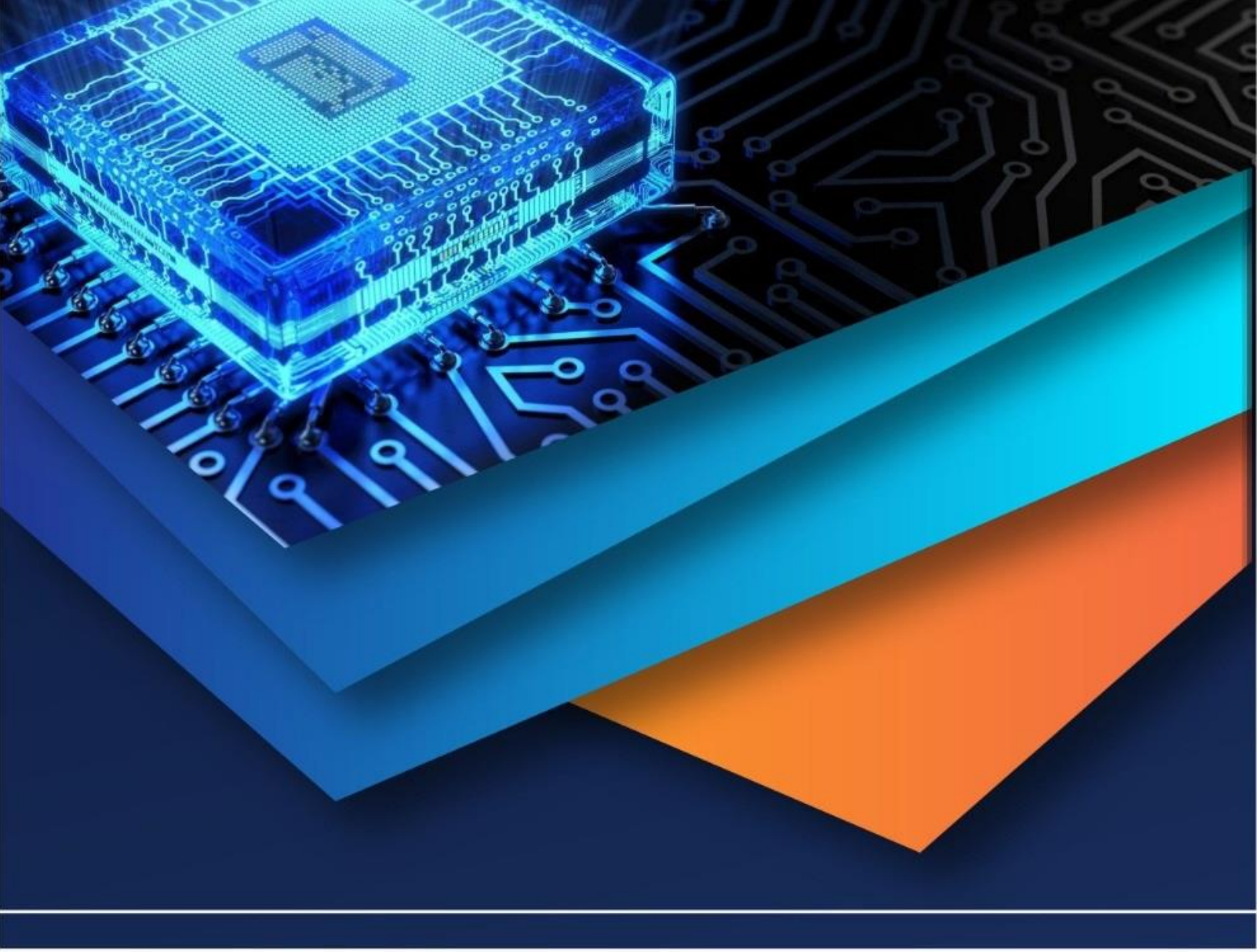
[3] Syed, N.F., Shah, S.W., Shaghaghi, A., Anwar, A., Baig, Z., Doss, R.: Zero Trust Architecture (ZTA): A Comprehensive Survey. IEEE Access 10, 57143–57179 (2022). https://doi.org/10.1109/ACCESS.2022.3174679

[4] MacQueen, J.: Some Methods for Classification and Analysis of Multivariate Observations. In: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297. University of California Press, Berkeley (1967).

[5] Lundberg, S.M., Lee, S.-I.: A Unified Approach to Interpreting Model Predictions. In: Advances in Neural Information Processing Systems 30 (NeurIPS 2017), pp. 4765–4774. Curran Associates, Long Beach (2017).

[6] Kounadi, O., Ristea, A., Araujo, A., Leitner, M.: A Systematic Review on Spatial Crime Forecasting. Crime Science 9, 1–22 (2020). https://doi.org/10.1186/s40163-020-00116-7

[7] Mushtaq, S., Mohsin, M., Mushtaq, M.M.: A Systematic Literature Review on the Implementation and Challenges of Zero Trust Architecture Across Domains. Sensors 25(19), 6118 (2025). https://doi.org/10.3390/s25196118



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)