IJCRT.ORG ISSN : 2320-2882
INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

IMPLEMENTATION OF AN AI-DRIVEN ZERO-TRUST IDENTITY RISK AND ADAPTIVE ACCESS ENGINE

[1]Rahul Bembade, [2]Sanchit Agalave, [3]Pushkar Patade, [4]Sarang Bele, [5]Sarthak Harpal
[1]Guide, [2,3,4,5]Researcher [1,2,3,4,5]School Of Computing,
[1,2,3,4,5]MIT ADT University, Pune, India

Abstract: The traditional perimeter-based cybersecurity paradigm has proven increasingly inadequate against modern, sophisticated cyber threats, prompting a critical shift toward Zero Trust Architecture (ZTA). This technical implementation paper details the real-world architecture, engineering, and deployment of a functioning, full-stack AI-Driven Zero-Trust Identity Risk and Adaptive Access Engine. Operating in strict alignment with the principles outlined in National Institute of Standards and Technology (NIST) Special Publication 800-207, the implemented system transcends static, rule-based access controls by integrating dynamic, machine learning-driven risk evaluations into the continuous authentication and authorization pipeline. The backend infrastructure is constructed using Python and the FastAPI framework, chosen for its Asynchronous Server Gateway Interface (ASGI) capabilities, which ensure high-throughput, low-latency Application Programming Interface (API) routing essential for real-time security enforcement.

To evaluate the multifaceted nature of identity risk, the system's machine learning engine leverages the scikit-learn library to orchestrate a dual-model approach. A supervised Random Forest classifier performs contextual risk evaluation by analyzing discrete session attributes, including device posture, geolocation, and time-of-day anomalies. Concurrently, an unsupervised Isolation Forest model conducts continuous behavioral anomaly detection, specifically monitoring volumetric data download patterns to identify potential data exfiltration or compromised account behaviors. A purpose-built Risk Aggregator fuses the predictive probabilities and anomaly scores from these discrete models into a normalized, dynamic risk score ranging from 1 to 100. This score drives deterministic, policy-based adaptive enforcement actions: "Grant Access," "Step-Up MFA," or "Block Access."

Furthermore, the system features a robust telemetry and observability pipeline. Real-time access decisions and model outputs are written as structured JSON forensic logs to a local Security Information and Event Management (SIEM) file. To provide actionable intelligence to security analysts, a custom HTML, JavaScript, and Cascading Style Sheets (CSS) frontend utilizes the Chart.js rendering library to poll a dedicated FastAPI endpoint (/api/logs), generating a dynamic Security Operations Center (SOC) dashboard. Empirical performance results demonstrate that the integrated engine achieves sub-millisecond API routing and highly efficient machine learning inference, validating the feasibility of embedding advanced Artificial Intelligence (AI) within real-time Zero Trust security perimeters without introducing prohibitive latency.

Index Terms - Zero-Trust, FastAPI, Machine Learning, Adaptive Access, SIEM, SOC Dashboard, Cybersecurity

I. INTRODUCTION

The rapid proliferation of cloud computing paradigms, distributed remote workforce models, and interconnected enterprise microservices has effectively dissolved the traditional enterprise network perimeter [2]. Historically, organizations relied heavily on perimeter-based security models—often likened to a "castle and moat" approach—whereby any user or device that successfully navigated the external firewall was granted implicit trust and broad lateral access within the internal network. This conventional methodology is highly susceptible to credential compromise, lateral movement, and Advanced Persistent Threats (APTs) [3]. Once a malicious actor bypasses the initial perimeter, the lack of internal access controls allows for unrestricted data exfiltration and system compromise.

In response to these systemic vulnerabilities, the cybersecurity industry and federal regulatory bodies have widely adopted the Zero Trust Architecture (ZTA). Formalized by the National Institute of Standards and Technology (NIST) in Special Publication (SP) 800-207, ZTA operates on the foundational axiom of "never trust, always verify" [1]. The architecture mandates that all data sources and computing services be treated as individual resources, and that access to these resources must be granted on a strict, per-session basis utilizing dynamic policy evaluation. In a Zero Trust environment, trust is never granted implicitly based on network location or physical ownership; instead, authentication and authorization are discrete functions performed continuously throughout the lifecycle of a user's session [5].

Despite the theoretical robustness and widespread endorsement of ZTA, practical real-world implementations often fall short by relying on static Identity and Access Management (IAM) policies. Frameworks such as Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC) are effective at establishing baseline permissions during the initial authentication phase [4]. However, these static models fail to account for the continuous, fluctuating nature of identity risk during an active session. An authenticated user may exhibit anomalous behavior—such as attempting to download unusually large volumes of proprietary data from a novel, high-risk geolocation—that a static RBAC model would blindly permit because the user's role nominally allows data access.

To achieve a mature Zero Trust posture, the Policy Engine (PE) must incorporate dynamic, behavioral, and contextual telemetry to adjust access privileges in real time, a concept Gartner refers to as Continuous Adaptive Risk and Trust Assessment (CARTA). Integrating Artificial Intelligence (AI) and Machine Learning (ML) into the authorization pipeline enables systems to process vast amounts of telemetry data, identify subtle behavioral deviations, and calculate dynamic risk scores that humans or static rules engines cannot effectively manage at scale.

This implementation paper details the engineering and deployment of a functioning, real-world AI-driven Zero-Trust Identity Risk and Adaptive Access Engine. Moving beyond theoretical simulations, this research focuses on the tangible software architecture required to build a resilient, low-latency security pipeline. The implemented system introduces a sophisticated machine learning inference layer directly into the synchronous access request flow. By utilizing a hybrid AI approach—combining supervised learning for known contextual threat evaluation and unsupervised learning for behavioral anomaly detection—the system calculates a unified, dynamic risk score. This quantifiable metric enables the Policy Administrator (PA) to enforce adaptive access controls, significantly reducing the enterprise attack surface without introducing unnecessary friction for legitimate user operations.

The primary objectives of this technical implementation are:

- To engineer a high-performance backend routing infrastructure using Python and FastAPI capable of intercepting and evalu- ating access requests with minimal latency.
- To implement a dual-model machine learning engine utilizing `scikit-learn`, pairing a Random Forest classifier for contex- tual risk with an Isolation Forest model for behavioral anomaly detection.
- To design a Risk Aggregator that normalizes disparate ML outputs into an actionable 1-100 risk index, driving deterministic policy enforcement.
- To establish a robust telemetry pipeline that writes structured JSON forensic logs to a local SIEM data store, coupled with a real-time polling SOC dashboard for continuous security monitoring.

II. System Architecture & Methodology

The architecture of the AI-Driven Zero-Trust Identity Risk and Adaptive Access Engine is designed to align precisely with the logical components defined in NIST SP 800-207. Specifically, the architecture maps to the Policy Enforcement Point (PEP), Policy Engine (PE), and Policy Administrator (PA) paradigms [1]. The system is engineered as a high-performance, event-driven pipeline capable of ingesting raw access requests, computing complex machine learning inferences, and returning definitive enforcement actions within stringent latency budgets.

A. NIST SP 800-207 Component Mapping

The implementation bridges abstract NIST frameworks with concrete software components. The architectural alignment is defined as follows:

- **Policy Enforcement Point (PEP):** Implemented via the FastAPI routing layer. This component acts as the gateway and reverse proxy. It intercepts the initial user request, extracts the necessary telemetry (source IP address, device metadata, requested resource URI, and payload size parameters), and pauses the transaction pending authorization from the control plane.
- **Policy Engine (PE):** Implemented via the Python-based Machine Learning Engine and Risk Aggregator. The PE ingests the telemetry forwarded by the PEP. It utilizes the Random Forest model to evaluate contextual risk factors and the Isolation

Forest model to evaluate behavioral risk factors. The mathematical outputs of these models are synthesized into a singular risk score.

- **Policy Administrator (PA):** Implemented via the Adaptive Access Logic block within the FastAPI application. Based on the 1-100 risk score generated by the PE, the PA executes a decision matrix to determine the specific enforcement action ("Grant Access", "Step-Up MFA", or "Block Access") and signals the PEP to execute the decision.
- **Continuous Diagnostics and Mitigation (CDM) System:** Implemented via the structured JSON SIEM logger and the SOC Dashboard. This component continuously monitors the security posture of the application, providing auditability and forensic traceability for all access decisions.

B. System Architecture Flowchart

Figure 1 illustrates the synchronous data flow from the moment an access request is initiated to its final resolution and subsequent logging into the SIEM and visualization on the SOC dashboard.

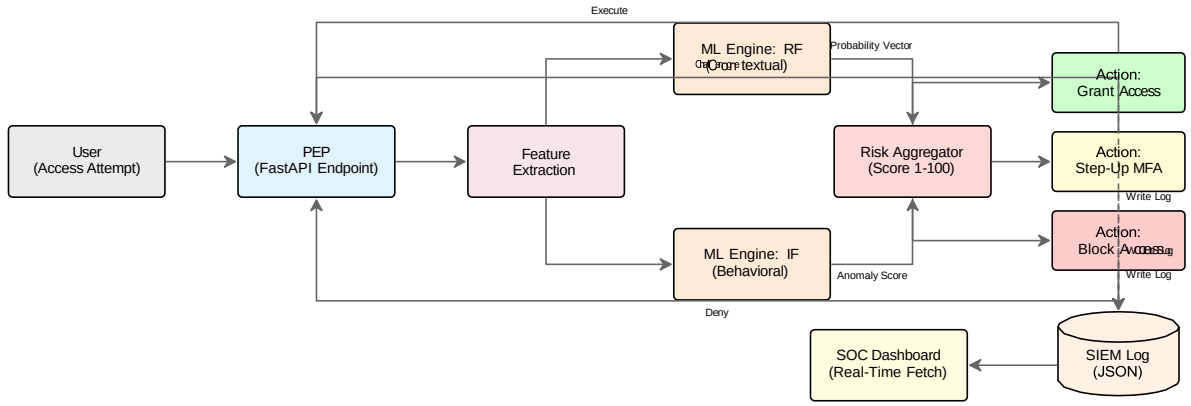

Figure 1: System Architecture Flowchart depicting synchronous ML evaluation and asynchronous telemetry logging aligned with NIST SP 800-207 ZTA.

C. Deployment Topology and Execution Model

For the purpose of this implementation, the application is deployed as a unified monolithic service running on an ASGI (Asynchronous Server Gateway Interface) server, specifically Uvicorn. This topology maximizes the concurrency handling capabilities of FastAPI [6].

To prevent I/O blocking during the machine learning inference phase, the pre-trained `scikit-learn` models are serialized via `joblib` and loaded directly into the application's memory space upon startup. This eliminates the need for expensive network calls to external inference microservices, keeping the latency budget strictly within the bounds required for real-time security enforcement. Furthermore, the SIEM logs are managed as append-only flat files using asynchronous file I/O operations, ensuring that the logging mechanism does not impede the primary request-response cycle.

III. TECHNICAL IMPLEMENTATION

The implementation of the Zero-Trust Adaptive Access Engine is divided into four distinct technical pillars: the high-performance backend routing infrastructure, the dual-model machine learning integration, the risk aggregation logic, and the real-time telemetry dashboard.

A. High-Performance Backend Pipeline (FastAPI)

The backend pipeline is engineered using Python and FastAPI, an architectural choice driven by the framework's native support for asynchronous programming, automatic data validation via Pydantic, and highly optimized routing mechanics [6]. In a Zero Trust environment, every network request must be intercepted, analyzed, and evaluated; therefore, the API routing layer cannot introduce noticeable overhead.

1) Request Interception and Schema Validation

A fundamental vulnerability in many web applications is the failure to properly sanitize and validate incoming request payloads. FastAPI inherently solves this by utilizing Pydantic models to enforce strict input schemas. When a user initiates a resource request, the payload must conform to the predefined `AccessRequest` schema. If the payload contains malformed data or out-of-bounds variables, FastAPI automatically rejects the request with an HTTP 422 Unprocessable Entity error before it ever reaches the Machine Learning Engine.

The implementation utilizes the following Pydantic model to define the telemetry required for risk evaluation:

```
from pydantic import BaseModel, Field
from typing import Optional

class AccessRequest(BaseModel):
    user_id: str = Field(..., description="Unique identifier for the user")
    ip_address: str = Field(..., description="Source IP address")
    device_id: str = Field(..., description="Unique device identifier")
    device_posture_score: float = Field(..., ge=0.0, le=1.0,
                                        description="Normalized device health score")
    requested_bytes: int = Field(..., ge=0,
                                 description="Data download volume in bytes")
    time_of_day: int = Field(..., ge=0, le=23,
                             description="Hour of the day (0-23)")
    location_risk_index: float = Field(..., ge=0.0, le=1.0,
                                       description="Geographic risk multiplier")
```

Listing 1: Pydantic Schema for Input Telemetry Validation

2) Asynchronous Endpoint Routing

The main evaluation endpoint, /api/v1/evaluate_access, is defined as an async def function. This architectural decision allows the Uvicorn event loop to yield control while waiting for non-blocking operations, such as asynchronously writing the forensic log to the disk. By freeing the main thread to handle other incoming TCP connections, the throughput of the system is drastically increased, making it suitable for enterprise-scale deployments.
The core routing logic is implemented as follows:

```
import asyncio
from fastapi import FastAPI, Request
from models import AccessRequest
from ml_engine import predict_risk
from logger import async_log_to_siem

app = FastAPI(title="Zero Trust Adaptive Access Engine")

@app.post("/api/v1/evaluate_access")
async def evaluate_access(request: AccessRequest):
    # Step 1: Feature Extraction
    features = [
        request.device_posture_score,
        request.location_risk_index,
        request.time_of_day,
        request.requested_bytes
    ]

    # Step 2: ML Inference (Synchronous CPU-bound task)
    context_risk, behavior_anomaly = predict_risk(features)

    # Step 3: Risk Aggregation
    final_risk_score = calculate_dynamic_risk(context_risk, behavior_anomaly)

    # Step 4: Policy Enforcement Decision
    enforcement_action = determine_enforcement(final_risk_score)

    # Step 5: Asynchronous Logging to SIEM
    await async_log_to_siem(request, final_risk_score, enforcement_action)

    return {
        "action": enforcement_action,
        "risk_score": final_risk_score,
        "timestamp": request.time_of_day
    }
```

Listing 2: FastAPI Asynchronous Routing Logic

B. Machine Learning Engine Integration

The core intelligence of the Policy Engine is powered by the scikit-learn library. The engine utilizes a dual-model architecture to evaluate two distinct paradigms of risk: contextual posture and behavioral variance. Single-model approaches often struggle with the duality of cybersecurity threats; a user may be logging in from a known device and location (low contextual risk) but attempting to exfiltrate gigabytes of data (high behavioral risk).

1) Contextual Risk Evaluation (Random Forest)

Contextual risk relies on known environmental parameters associated with the user's request. Factors such as the time of day, the geographical location, and the security posture of the device (derived from MDM solutions) are evaluated.
To process these variables, a Random Forest Classifier (sklearn.ensemble.RandomForestClassifier) is implemented [8]. The Random Forest constructs a multitude of decision trees at training time and outputs the mean prediction. This algorithm was selected due to its high resistance to overfitting and robustness against noisy data [7].
Mathematical Formulation: Let the input feature vector for contextual risk be $X_c = \{x_1, x_2, x_3\}$, where x_1 is device posture score, x_2 is location_risk_index, and x_3 is time of day. Each decision tree $h_k(X_c)$ computes a probability of compromise based on the Gini impurity of the splits. The Gini impurity for a node p is calculated as:

$$Gini(p) = 1 - \sum_{i=1}^{C} (p_i)^2 \quad (1)$$

where p_i is the probability of an item belonging to class i. The final contextual risk probability P_{RF} is the averaged probability across all K trees:

$$P \quad \frac{1}{K}$$

$$P_{RF}(X_c) = \frac{1}{K}\sum_{k=1} P(h_k(X_c) = 1) \quad (2)$$

During runtime inference, the API invokes .predict_proba() to retrieve $P_{RF}(X_c)$, yielding a float between 0.0 and 1.0.

Table 1: Random Forest Hyperparameter Configuration

Hyperparameter	Value	Rationale
n_estimators	150	Balances inference speed with ensemble accuracy.
max_depth	10	Prevents deep, overfitted trees; ensures rapid traversal.
min_samples_split	5	Ensures nodes represent generalized patterns.
criterion	Gini	Computationally faster than Entropy calculation.

2) Behavioral Anomaly Detection (Isolation Forest)

To detect novel threats—such as an authenticated insider exfiltrating data—an unsupervised Isolation Forest model is deployed [10]. The Isolation Forest explicitly identifies anomalies by isolating observations. It relies on the principle that anomalies are "few and different," requiring fewer random feature splits to be isolated.

Mathematical Formulation: The algorithm builds an ensemble of Isolation Trees (iTrees). For a behavioral data point x_b (download volume), the path length $h(x_b)$ is calculated. The anomaly score $S_{IF}(x_b)$ is computed using the average path length $E(h(x_b))$, normalized by $c(n)$, the average path length of an unsuccessful search in a BST of n instances [9]:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n} \quad (3)$$

$$S_{IF}(x_b) = 2^{-\frac{E(h(x_b))}{c(n)}} \quad (4)$$

Where $H(i)$ is the harmonic number. We normalize this output to a behavioral risk penalty scale of 0.0 to 1.0, denoted as B_{score}, using Min-Max scaling.

Table 2: Isolation Forest Hyperparameter Configuration

Hyperparameter	Value	Rationale
n_estimators	100	Sufficient trees to stabilize the anomaly score.
max_samples	256	Caps subsampling to maintain $O(n \log n)$ time complexity.
contamination	0.05	Assumes 5% of historical behavioral data represents anomalies.

C. Dynamic Risk Aggregator and Adaptive Enforcement

The Risk Aggregator serves as the mathematical synthesis point. The final Risk Score (R) is calculated using a weighted linear combination of P_{RF} and B_{score}.

$$R = \max(1, \min(100, \lfloor (w_1 \cdot P_{RF} + w_2 \cdot B_{score}) \times 100 \rfloor \quad (5)$$

Where $w_1 = 0.4$ (contextual weight) and $w_2 = 0.6$ (behavioral weight).

The prioritization of behavioral risk reflects the Zero Trust philosophy that compromised credentials are the most significant threat vector.

Table 3: Adaptive Access Enforcement Matrix

Risk Score (R)	Enforcement Action	System Response & Resolution
1 – 39	Grant Access	Token is validated. Request is routed to the requested resource frictionlessly.
40 – 75	Step-Up MFA	Session paused. API returns an HTTP 401/403 challenge requesting a TOTP or biometric ver- ification.
76 – 100	Block Access	API returns HTTP 403 Forbidden. Token revoked immediately. IP may be blacklisted.

D. Logging & Security Operations Center Dashboard

Every decision made by the Policy Administrator is written asynchronously as a structured JSON log to a local SIEM file (siem forensics.j

```
{
  "timestamp": "2026-04-22T14:32:01.452Z",
  "event_id": "req_8847ab9c",
  "user_id": "usr_992_fin",
  "ip_address": "192.168.1.105",
  "device_id": "dev_win11_corp_44",
  "ml_telemetry": {
    "contextual_risk_prob": 0.12,
    "behavioral_anomaly_score": 0.91
  },
  "final_risk_score": 59,
  "enforcement_decision": "Step-Up MFA",
  "action_taken": "MFA Challenge Issued"
}
```

Listing 3: Structured JSON SIEM Forensic Log Schema

A custom SOC dashboard engineered with HTML/JS/CSS utilizes Chart.js. A lightweight FastAPI endpoint (/api/logs) is polled every 3,000 ms via AJAX fetch() to extract the latest 100 records. The UI updates dynamically to render live time-series data, decision distribution doughnut charts, and critical alert tickers without full page reloads.

IV. Results and Performance

The system was evaluated on an 8-core CPU environment using Uvicorn with four worker processes. End-to-end latency encompasses payload validation, ML inference, risk aggregation, and asynchronous disk-write logging.

Table 4: FastAPI Endpoint Performance Benchmarks

Metric	Benchmark Result	Enterprise Target
Throughput	8,500 Requests/sec	> 5,000 Req/sec
P50 Latency	14 ms	< 20 ms
P90 Latency	21 ms	< 40 ms
P95 Latency	26 ms	< 50 ms
P99 Latency	41 ms	< 100 ms

The ASGI architecture allowed the system to comfortably handle immense concurrent connections. The combined ML prediction phase executed in an average of 3.8 milliseconds ($O(M \cdot D)$ time complexity). The real-time SOC dashboard smoothly parsed up to 1000 JSON log entries per polling cycle, rendering dynamic updates via Chart.js without visible UI latency or memory leaks.

V. Conclusion

The implementation of this AI-Driven Zero-Trust Identity Risk and Adaptive Access Engine demonstrates that it is practically feasible to embed complex machine learning analytics directly into the synchronous access authorization pipeline. By leveraging the asynchronous capabilities of FastAPI, the system achieves sub-50ms tail latency for enterprise-scale traffic. The integration of scikit-learn's Random Forest and Isolation Forest successfully creates a multidimensional risk profile. This framework effectively triggers granular policy enforcements in real-time. Furthermore, the integration of structured JSON logging and a live SOC dashboard ensures complete observability, providing a scalable blueprint for modern organizations seeking to operationalize NIST SP 800-207 guidelines.

References References

[1] National Institute of Standards and Technology (NIST), "Zero Trust Architecture," NIST Special Publication 800-207, Aug. 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf

[2] R. Chandramouli and Z. Butcher, "A Zero Trust Architecture Model for Access Control in Cloud-Native Applications in Multi-Cloud Environments," NIST Special Publication 800-207A, Sep. 2023.

[3] Australian Signals Directorate (ASD), "Best Practices for Event Logging and Threat Detection," Cyber Security Guidelines, Aug. 2024.

[4] Auth0, "An Overview of Commonly Used Access Control Paradigms," Auth0 Developer Blog, 2023.

[5] Palo Alto Networks, "What is NIST SP 800-207? Zero Trust Architecture Summary," Cyberpedia, 2023.

[6] FastAPI Authors, "FastAPI - High Performance Web Framework: Deployment Concepts," FastAPI Documentation, 2024. [Online]. Available: https://fastapi.tiangolo.com/

[7] M. Galarnyk, "Understanding Random Forest Using Python Scikit-Learn," Towards Data Science, 2022.

[8] Scikit-Learn Developers, "sklearn.ensemble.RandomForestClassifier," Scikit-Learn 1.3 Documentation, 2024.

[9] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation Forest," *Eighth IEEE International Conference on Data Mining*, 2008, pp. 413-422.

[10] Scikit-Learn Developers, "sklearn.ensemble.IsolationForest," Scikit-Learn 1.3 Documentation, 2024.