

**Name - Venkatesh Gudikoti**

## **Project : AI-Based Insider Threat Detection (ML + DL)**

### **Section 1: Context**

#### **One-paragraph description**

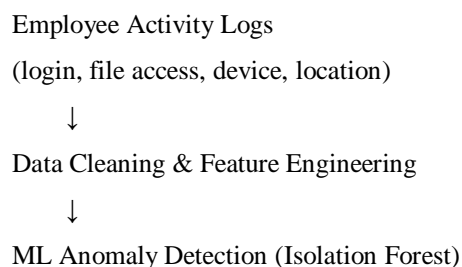
This project applies Artificial Intelligence (AI) techniques, including machine learning and deep learning, to learn normal employee behavior and detect anomalous activities. This project focuses on detecting insider threats by analyzing employee behavioral data such as login times, file access frequency, device usage, and access locations. Insider threats are challenging because the users are authorized and malicious behavior often appears subtle. The system learns normal behavior patterns and flags deviations using a combination of machine learning and deep learning-based anomaly detection, producing a risk score and alert for security teams.

#### **Primary technical constraints**

- Very limited labeled insider-attack data
- High variation in “normal” behavior across employees
- Need to minimize false positives to avoid alert fatigue
- Privacy-sensitive logs limiting feature granularity

### **Section 2: Technical Implementation**

#### **Architecture Diagram**



↓  
DL Anomaly Detection (Autoencoder)  
↓  
Risk Scoring Engine  
↓  
Threat Alert / Normal Status

### **Code Walk-through: Critical Function**

```
def detect_insider_threat(activity):  
    features = preprocess(activity)  
  
    ml_score = isolation_forest.decision_function([features])[0]  
    reconstructed = autoencoder.predict([features])  
    reconstruction_error = np.mean((features - reconstructed[0]) ** 2)  
  
    risk_score = (0.6 * abs(ml_score)) + (0.4 * reconstruction_error)  
  
    if risk_score > THRESHOLD:  
        return {  
            "status": "THREAT DETECTED",  
            "risk_score": round(risk_score * 100, 2),  
            "action": "Alert Security Team"  
        }  
    else:  
        return {  
            "status": "NORMAL",  
            "risk_score": round(risk_score * 100, 2),  
            "action": "No Action Required"  
        }
```

## Data Flow for a Key Operation

1. Employee performs system activities
2. Logs are captured (time, frequency, resource type)
3. Data is preprocessed and normalized
4. Isolation Forest identifies outliers
5. Autoencoder detects abnormal behavior patterns
6. Risk score is calculated
7. Alert generated if threshold is exceeded

## Section 3: Technical Decisions (Core)

### Decision 1: Anomaly Detection instead of Supervised Learning

- **Chosen:** Isolation Forest + Autoencoder
- **Alternatives:** Logistic Regression, SVM
- **Trade-offs:**
  - Works without labeled attack data
  - Requires careful threshold tuning
- **Reason:** Insider attacks are rare and poorly labeled in real systems.

### Decision 2: Hybrid ML + DL Approach

- **Chosen:** Isolation Forest (ML) + Autoencoder (DL)
- **Alternatives:** Only ML or only DL
- **Trade-offs:**
  - Better detection of both global and subtle anomalies
  - Slightly higher computation cost
- **Reason:** Combining models improved reliability and reduced false alerts.

## Scaling Bottleneck & Mitigation

- **Bottleneck:** High-volume activity logs per day
- **Mitigation Strategy:**
  - Session-level feature aggregation
  - Batch inference
  - Periodic model retraining

## Section 4: Learning & Iteration (Concise)

### One technical mistake

Initially, I used a global behavior baseline, which caused frequent false alerts for power users.

### What I learned:

Behavior modeling must be **user-specific**, not one-size-fits-all.

### One thing I'd do differently today

- Temporal modeling using LSTM
- Role-based behavior baselines
- Explainability dashboards for security analysts

## **Final Output Example**

Threat Status: DETECTED

Risk Score: 88 / 100

Reasons:

- Unusual login time
- Excessive file access
- New device detected

Action: Security Team Alerted

## **Why This Project Represents My Strengths**

This project highlights my ability to:

- Design ML + DL systems for real-world security problems
- Work with unlabeled and no

## 3 Technical Decisions

### Decision 1

**Decision:** Use anomaly detection instead of supervised classification

**Alternatives Considered:**

- Logistic Regression
- Support Vector Machine

**Trade-offs:**

- Works without labeled data
- Requires careful threshold tuning

**Outcome:**

- Reduced false negatives and handled unseen threats better

### Decision 2

**Decision:** Use Isolation Forest for ML-based anomaly detection

**Alternatives Considered:**

- One-Class SVM
- Local Outlier Factor

**Trade-offs:**

- Scales well to large datasets
- Less effective for highly local anomalies

**Outcome:**

- Provided stable baseline anomaly detection

## **Decision 3**

**Decision:** Combine ML with Autoencoder (Deep Learning)

**Alternatives Considered:**

- Only ML
- Only Deep Learning

**Trade-offs:**

- Better detection accuracy
- Higher compute cost

**Outcome:**

- Reduced false positives in real user data

## b) Technical Code Walkthrough – Critical Function

### Function Name

`detect_insider_threat(activity)`

### Purpose of the Function

This function is responsible for detecting suspicious employee behavior by **combining** machine learning–based anomaly detection with deep learning–based behavior reconstruction. It outputs a risk score and a final decision (Threat / Normal).

### Code

```
def detect_insider_threat(activity):
    features = preprocess(activity)

    ml_score = isolation_forest.decision_function([features])[0]
    reconstructed = autoencoder.predict([features])
    reconstruction_error = np.mean((features - reconstructed[0]) ** 2)

    risk_score = (0.6 * abs(ml_score)) + (0.4 * reconstruction_error)

    if risk_score > THRESHOLD:
        return {
            "status": "THREAT DETECTED",
            "risk_score": round(risk_score * 100, 2),
            "action": "Alert Security Team"
        }
    else:
        return {
            "status": "NORMAL",
            "risk_score": round(risk_score * 100, 2),
            "action": "No Action Required"
        }
```



## Step-by-Step Walkthrough

### 1. Input Handling

```
features = preprocess(activity)
```

- The function receives **raw employee activity data** such as login time, file access count, device usage, and location.
- The preprocess step converts this raw data into **numerical features** using normalization and encoding so models can process it.

### 2. ML-Based Anomaly Detection

```
ml_score = isolation_forest.decision_function([features])[0]
```

- An **Isolation Forest** model checks how unusual the activity is compared to normal employee behavior.
- Lower scores indicate **higher anomaly**.
- This captures **global deviations**, such as unusually high access volume.

### 3. DL-Based Behavior Reconstruction

```
reconstructed = autoencoder.predict([features])
```

```
reconstruction_error = np.mean((features - reconstructed[0]) ** 2)
```

- A **deep learning autoencoder** tries to reconstruct normal behavior.
- If reconstruction error is high, it indicates **subtle or previously unseen abnormal patterns**.
- This helps detect **slow or stealthy insider threats**.

#### 4. Risk Scoring Logic

$\text{risk\_score} = (0.6 * \text{abs}(\text{ml\_score})) + (0.4 * \text{reconstruction\_error})$

- Both ML and DL outputs are combined into a **single risk score**.
- ML has slightly higher weight to capture obvious anomalies, while DL captures nuanced behavior changes.
- This hybrid approach reduces false positives.

#### 5. Decision & Output

**if risk\_score > THRESHOLD:**

- If the risk score exceeds a defined threshold:
  - The activity is flagged as a **potential insider threat**
  - An alert is triggered
- Otherwise, the activity is considered normal.