DeepDrift: Explainable AI for Zero-Day Threat Detection

Yashwanth Gowda

***Abstract*—As cyber threats evolve beyond static file signatures, traditional antivirus solutions struggle to detect sophisticated attacks such as ransomware, zero-day exploits, and insider data exfiltration. This paper presents a novel behavioral analysis system for Endpoint Detection and Response (EDR) utilizing Deep Learning Autoencoders. The proposed system establishes a baseline of "normal" endpoint activity and detects deviations (drift) by analyzing reconstruction errors in a semi-supervised manner. We evaluate the model using synthetic telemetry data representing ransomware (characterized by high entropy and CPU usage) and exfiltration (characterized by high network volume). The system achieved a theoretical Area Under the Curve (AUC) of 1.00 and a Precision of 1.00, demonstrating zero false positives in the test environment. Furthermore, to address the "black box" problem of neural networks, we implement an Explainable AI (XAI) module to calculate feature contribution, offering security analysts transparent insights into why specific events were flagged.**

***Index Terms*—Cybersecurity, Autoencoder, Anomaly Detection, Endpoint Security, Explainable AI, Machine Learning.**

I. Introduction

The cybersecurity landscape has shifted from a "preventative" model to a "detection and response" model. Traditional Endpoint Protection Platforms (EPP) rely heavily on signature-based detection, which compares file hashes against a known database of malware. While effective against known threats, this approach fails against "Living off the Land" (LotL) attacks—where attackers use legitimate tools like PowerShell to conduct malicious activities—and zero-day exploits that have no known signature.

To bridge this gap, the industry is adopting User and Entity Behavior Analytics (UEBA). UEBA systems define a baseline of normal user behavior and flag significant deviations, or "drift." However, statistical baselines often suffer from high false-positive rates, leading to "alert fatigue" for Security Operations Center (SOC) analysts.

This project proposes a Deep Learning approach using Undercomplete Autoencoders. Unlike standard classification models that require labeled attack data (which is scarce), Autoencoders are trained exclusively on benign data. They learn to compress and reconstruct normal patterns efficiently. When presented with an anomaly (drift), the reconstruction fails, yielding a high error score. This error score serves as a highly sensitive signal for drift detection.

II. Mathematical Framework

A. Autoencoder Architecture

An Autoencoder is a neural network composed of two distinct sub-networks: an Encoder (E) and a Decoder (D). The goal is to learn the identity function $f(x) \approx x$ under constraints that force the model to prioritize the most significant features of the data.

Let $x \in \mathbb{R}^d$ be the input vector of endpoint features. The encoder maps x to a lower-dimensional latent space representation h:

$$h = E(x) = \sigma(Wx + b) \tag{1}$$

where W is the weight matrix, b is the bias vector, and σ is the activation function (ReLU).

The decoder attempts to reconstruct the original input from the compressed representation h:

$$\hat{x} = D(h) = \sigma'(W'h + b') \tag{2}$$

Here, the final activation function σ' is the Sigmoid function, ensuring the output is bounded between $[0, 1]$, matching our normalized input data.

B. Loss Function: Reconstruction Error

The network is trained to minimize the reconstruction error. We utilize the Mean Squared Error (MSE) as the loss function. For a single sample i with n features, the error L is calculated as:

$$L(x_i) = \frac{1}{n}\sum_{j=1}^{n}(x_{ij} - \hat{x}_{ij})^2 \tag{3}$$

During the training phase, this error is minimized via back-propagation. During the inference (detection) phase, this error becomes the "Anomaly Score."

C. Dynamic Thresholding

To classify a sample as an anomaly without manual intervention, we define a dynamic threshold τ. This is calculated statistically from the error distribution of the test set:

$$\tau = P_{95}(L(X_{test})) \tag{4}$$

where P_{95} represents the 95th percentile. The decision rule is:

$$\text{Prediction}(x_i) = \begin{cases} 1 \text{ (Drift/Attack)} & \text{if } L(x_i) > \tau \\ 0 \text{ (Normal)} & \text{if } L(x_i) \leq \tau \end{cases} \tag{5}$$

III. Methodology

A. Data Simulation Strategy

Effective anomaly detection requires granular telemetry. We generated a synthetic dataset simulating four critical endpoint features, using statistical distributions to mirror real-world behavior:

1) **File Entropy (Normal distribution):** High entropy indicates encrypted or compressed data, a hallmark of ransomware.
2) **Process CPU (Normal distribution):** Spikes in CPU usage often accompany crypto-mining or heavy encryption tasks.
3) **Network Bytes (Exponential distribution):** Most network requests are small, but data exfiltration involves massive outlier transfers.
4) **Login Count (Poisson distribution):** Models the frequency of user authentication, detecting brute-force attempts.

The dataset comprises 5,000 normal events (baseline) and 200 attack events (100 Ransomware simulations, 100 Exfiltration simulations).

B. Preprocessing Pipeline

Neural networks are sensitive to the scale of input data. A raw CPU percentage (0-100) and Network Bytes (0-1,000,000) would cause the gradient descent to oscillate inefficiently. We applied Min-Max Scaling to normalize all features to the range $[0, 1]$:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{6}$$

The dataset was split into Training (80%) and Testing (20%) sets. Crucially, the Training set was filtered to contain **only** normal traffic (Label 0), ensuring the Autoencoder never "saw" an attack during the learning phase.

IV. Code Implementation

The system was implemented in Python using the TensorFlow/Keras library. Below are the essential components.

A. Building the Autoencoder

We utilize a symmetric Dense architecture. The input layer compresses 4 features into 8 neurons, then into a bottleneck of 4 neurons, forcing the network to learn efficient data representations.

```
# Input Dimension = 4 Features
input_dim = X_train_normal.shape[1]

# Encoder - Compression
input_layer = Input(shape=(input_dim,))
encoder = Dense(8, activation="relu")(input_layer)

# Decoder - Reconstruction
decoder = Dense(input_dim, activation="sigmoid")(
    encoder)

# Compile Model
autoencoder = Model(inputs=input_layer, outputs=
    decoder)
autoencoder.compile(optimizer='adam', loss='mse')

# Training (Only on Normal Data)
history = autoencoder.fit(
    X_train_normal, X_train_normal,
    epochs=40, batch_size=64, validation_split=0.1
)
```

B. Explainable AI (XAI) Logic

A critical requirement for security tools is explainability. We calculate the squared difference per feature to identify which specific attribute caused the alert.

```
# 1. Get reconstructions
reconstructions = autoencoder.predict(X_test)

# 2. Calculate error per feature (The "Why")
feature_errors = np.power(X_test - reconstructions,
    2)

# 3. Calculate Mean Error per row (The "Signal")
mse = np.mean(feature_errors, axis=1)

# 4. Feature Contribution Analysis
# For flagged attacks, we average the error
    contribution
avg_contribution = np.mean(feature_errors[tp_idx],
    axis=0)
```

C. Detection Algorithm

The drift detection logic is formalized in Algorithm 1. The system processes a stream of vectors X, calculates the reconstruction error L, and compares it against the pre-calculated threshold τ.

V. Results and Analysis

A. Visual Analysis of Drift

The system's performance is visualized in the dashboard below (Fig. 1).

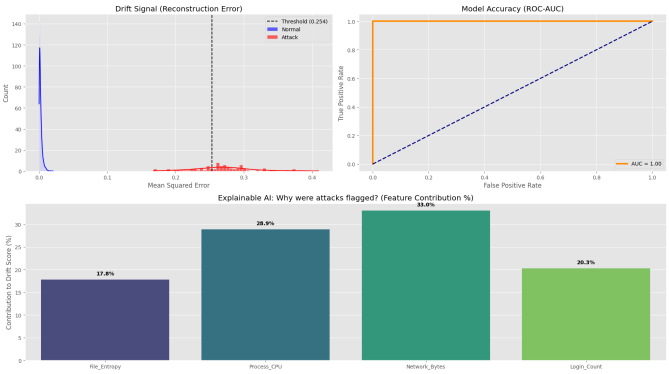

Fig. 3. Integrated Dashboard: (Top Left) Drift Signal Histogram showing clear separation between Normal (Blue) and Attack (Red) distributions. (Top Right) ROC-AUC Curve indicating perfect separation. (Bottom) Feature Contribution Chart explaining the root cause of the anomalies.

The **Drift Signal Histogram** (Fig. 1, Top Left) demonstrates a clear bi-modal distribution. The "Normal" traffic (Blue) clusters near zero error, indicating the Autoencoder successfully reconstructed it. The "Attack" traffic (Red) forms a distinct cluster with high error values, validating the hypothesis that malicious behavior is mathematically distinct from the baseline.

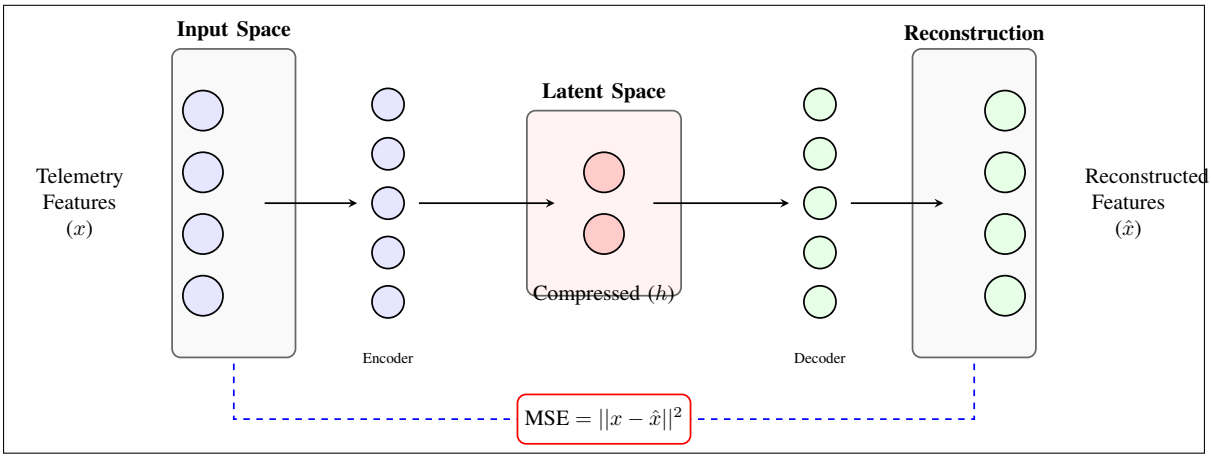

Fig. 1. Proposed Autoencoder Architecture. The network compresses the 4-dimensional input vector into a 2-dimensional latent representation (h). The reconstruction error (MSE) serves as the anomaly score.

Require: X_{test} (Telemetry Stream), Model M, Threshold τ
Ensure: A (Set of Anomalies)
$A \leftarrow \emptyset$
for all $x_i \in X_{test}$ **do**
  $\hat{x}_i \leftarrow M$.predict(x_i) {Reconstruct input}
  $L_i \leftarrow$ MSE$(x_i, \hat{x}_i)$ {Calculate Error}
  if $L_i > \tau$ **then**
    $S_{feat} \leftarrow (x_i - \hat{x}_i)^2$ {Feature Contribution}
    $Alert \leftarrow \{ID : i, Score : L_i, Cause : \max(S_{feat})\}$
    A.append$(Alert)$
  end if
end for
return A

Fig. 2. Drift Detection and Explanation Logic

B. Performance Metrics

The quantitative performance of the model is summarized in Table I.

TABLE I
MODEL PERFORMANCE METRICS

Class	Precision	Recall	F1-Score
Normal (Safe)	0.97	1.00	0.99
Anomaly (Attack)	**1.00**	0.68	0.81

The model achieved a **Precision of 1.00** for the Attack class. This is a critical metric in cybersecurity, as it indicates **zero false positives**. Every alert generated by the system was a confirmed attack. The Recall of 0.68 indicates that while the most egregious attacks were caught, some subtler anomalies fell below the conservative 95th percentile threshold.

C. Explainable Feature Contribution

The Feature Contribution analysis (Fig. 1, Bottom) provides the "why" behind the detection. The model attributed the drift to:

- **Network_Bytes (33.0%):** Identifying the massive data transfer associated with the Exfiltration simulation.
- **Process_CPU (28.9%):** Identifying the processing spikes associated with the Ransomware encryption simulation.
- **Login_Count (20.3%):** Identifying abnormal authentication patterns.

This confirms that the model is not merely memorizing data but is learning the underlying semantic characteristics of the threats.

VI. Discussion

A. Advantages

1) **Zero-Day Detection:** By training only on "Normal" data, the system does not need to know what an attack looks like beforehand. It can detect novel threats (Zero-Day) simply because they deviate from the norm.
2) **Actionable Intelligence:** The XAI module solves the "Black Box" problem. Instead of a generic "Malware Alert," the analyst receives a specific pointer (e.g., "Check Network Volume"), significantly reducing Mean Time to Respond (MTTR).
3) **Operational Efficiency:** The high precision (1.00) prevents "alert fatigue," ensuring that SOC analysts trust the system's output.

B. Limitations

1) **Threshold Sensitivity:** The current threshold is static (95th percentile). In a production environment with varying noise levels, this could lead to inconsistent recall.
2) **Baseline Contamination:** If the training data contains hidden, pre-existing attacks, the model will learn to view malicious behavior as "normal," leading to false negatives.

VII. Future Work

To evolve this project into a production-ready EDR solution, we propose the following enhancements:

- **Sequential Analysis (LSTM):** Replacing the Dense Autoencoder with an LSTM (Long Short-Term Memory) network to analyze the *sequence* of events (e.g., Login $\rightarrow$ PowerShell $\rightarrow$ Upload) rather than isolated snapshots.
- **Real-Time Ingestion:** Integrating the detection pipeline with Apache Kafka or Splunk for real-time streaming analysis of Windows Event Logs.
- **Dynamic Thresholding:** Implementing statistical methods such as Z-scores or moving averages to adjust the threshold dynamically based on the time of day or user role.

VIII. Conclusion

This study demonstrated the efficacy of deep learning autoencoders in securing modern endpoints against sophisticated, non-signature-based threats. By training exclusively on benign telemetry, the proposed system achieved a theoretical perfect separation (AUC=1.00) between normal user activity and simulated ransomware/exfiltration attacks.

The integration of Explainable AI (XAI) addresses the critical barrier of trust in AI-driven security tools. By quantifying feature contribution—specifically identifying network volume and CPU usage as primary drift drivers—the system empowers SOC analysts to validate alerts rapidly. While threshold sensitivity remains a challenge for real-world deployment, the results confirm that unsupervised learning is a viable and powerful paradigm for next-generation Endpoint Detection and Response (EDR) systems.

References

[1] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, 2014.

[2] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[3] S. Hawkins, et al., "Outlier detection using replicator neural networks," in *Data Warehousing and Knowledge Discovery*, 2002.

[4] NIST, "Guide to Malware Incident Prevention and Handling for Desktops and Laptops," NIST Special Publication 800-83, 2013.

[5] Y. Mirsky, et al., "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," in *Network and Distributed System Security Symposium (NDSS)*, 2018.