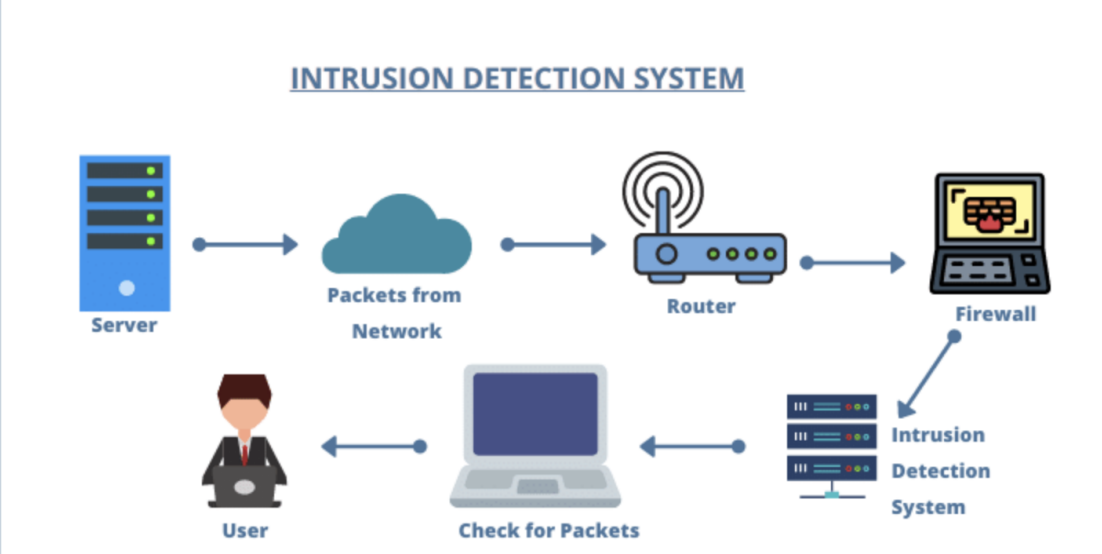
Cybersecurity Intrusion Detection System (IDS)



Course: BUAN 6341 – Applied Machine Learning

Group 4

- Shriya Reddy Kolan
- Vidhya Bharati Kulkarni
- Mehul Singh Rajaputhra
- Riddhima Reddy Ramasahayam
- Tara Canugovi

Introduction

In this project, we are building a machine learning-driven Intrusion Detection System (IDS) to identify and flag potential cybersecurity threats based on session-level data from network activity logs.

The project uses both:

- Supervised learning (with labeled attack data) to classify known intrusion
- Unsupervised learning (without labels) to detect novel, unknown threats via anomaly detection

Problem Statement

Traditional rule-based security systems are reactive and signature-dependent—they can only detect known attacks. In contrast, modern cyber threats are:

- Polymorphic (constantly changing)
- Subtle (blending in with legitimate traffic)
- Often undiscovered until after significant damage is done

As organizations process millions of network sessions daily, manual monitoring becomes infeasible. Machine learning offers a scalable, intelligent solution that can:

- Automatically learn threat patterns
- Detect behavioral anomalies
- Adapt over time to new threats

Exploratory Data Analysis

Dataset Dimensions

- **Total Records**: 9,537 session
- **Total Features**: 11 columns
- **CIC-IDS Dataset**
  - **Provided by:** Canadian Institute for Cybersecurity

Features
session_id
network_packet_size
protocol_type
login_attempts
session_duration
encryption_used
ip_reputation_score
failed_logins
browser_type
unusual_time_access
attack_detected

Handling Missing Values

Updated Value Counts:

Encryption Type	Count
AES	4,706
DES	2,865
None	1,966

Only one feature contained missing values

Encryption used had **1,966 missing entries** out of 9,537 records.

We **did not drop** these records or impute them with the mode (AES), because:

- That would assume encryption was used, possibly **masking risky behavior**
- Mode imputation introduces bias, especially when **missingness is non-random**

Strategy Applied:

Replaced missing values with 'None'

This explicitly captures the idea of **unencrypted or unknown encryption sessions**

Oversampling Rare Cases

Data cleaning –

We have dropped the session_id

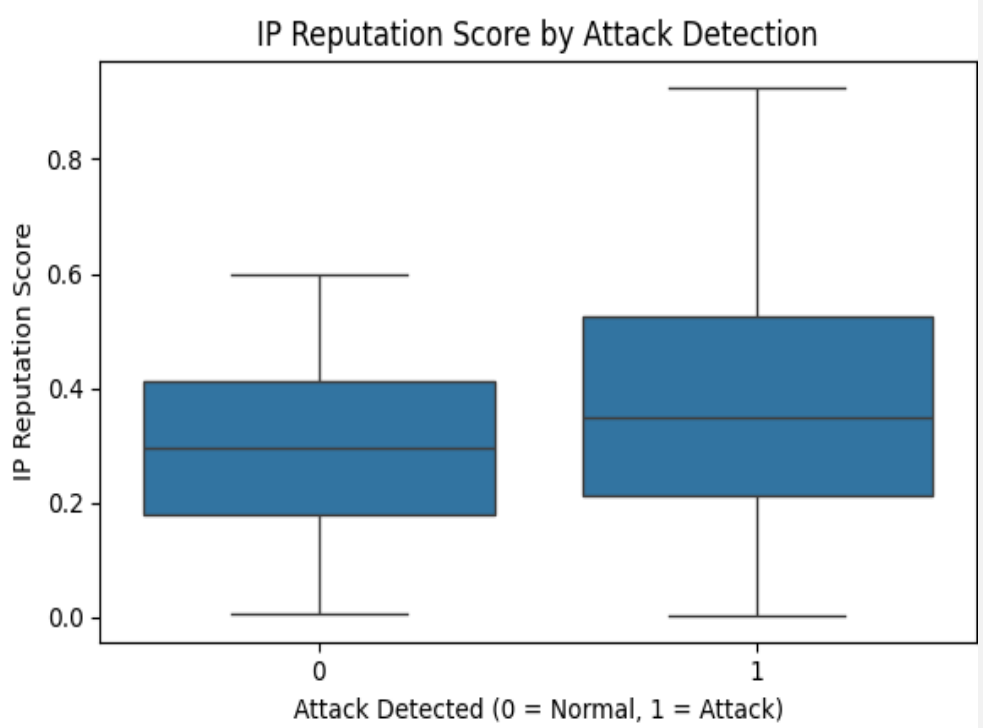
Why **Oversampling** in Cybersecurity ?

Models need **enough examples of intrusions** to understand what they look like

Without enough samples, ML will **ignore** rare attack patterns

Oversampling gives the model a **better chance to generalize** threat behavior

Insights from Visualization



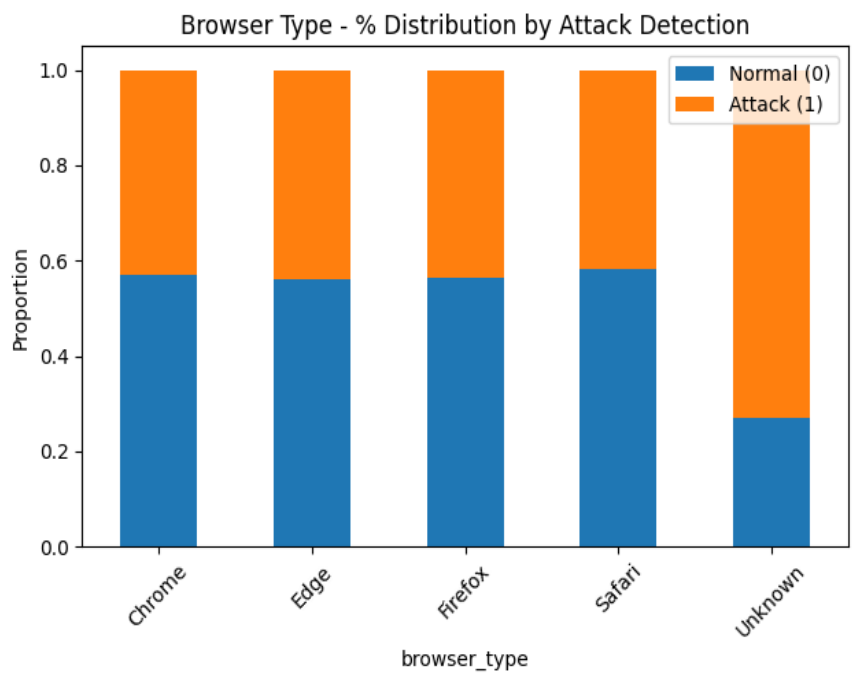
Interpretation:

- **Normal sessions** mostly have IP scores in the range of **0.1 to 0.6**.
- **Attack sessions** show a **wider and more varied score distribution** — ranging from very low (~0.01) to quite high (~0.9+).
- The **median score** is slightly higher for attacks than for normal sessions.

Cybersecurity Insight:

- Attack sessions originate from a more diverse set of IP addresses

Insights from Visualization



Cybersecurity Interpretation

Malicious actors often **spoof or strip headers** to avoid detection, resulting in an undefined User-Agent.

The use of "Unknown" browsers is a **red flag** for intrusion detection models and should be treated as a **high-risk category**.

This feature could be encoded categorically or even used to **trigger rule-based alerts** in real-time systems.

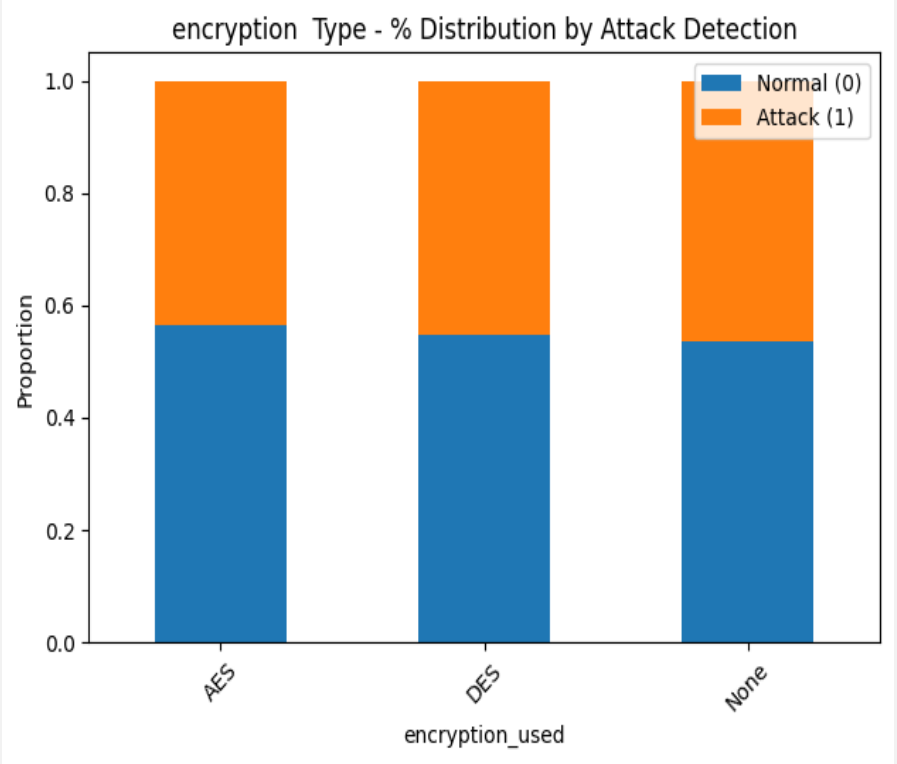
Insights from Visualization

Interpretation:

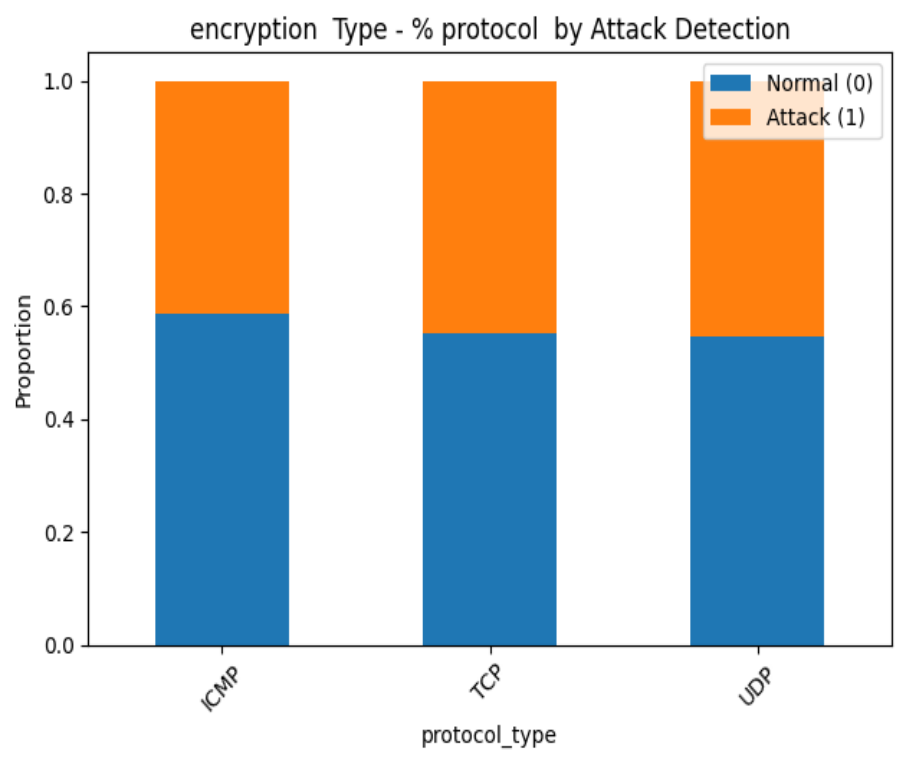
- Attackers may be using **encryption just as frequently as benign users**, possibly to **evade detection** or mimic legitimate traffic.
- The **lack of clear separation** suggests:
- Encryption type **alone is not predictive** of attacks in this dataset.
- It likely **needs to be combined** with other features (e.g., protocol type, IP reputation) for meaningful signals.

Cybersecurity Insight:

- Encryption type shows no strong correlation with attack likelihood — attackers seem to use encryption similarly to benign users, limiting its predictive power in isolation.



Insights from Visualization



Interpretation:

- Attackers prefer not to use any specific protocol in this dataset.
- This suggests that protocol_type is a low-signal feature for classification on its own.

Cybersecurity Insight:

- While certain protocols are commonly abused in real-world attacks (e.g., UDP for DDoS, ICMP for scanning), this dataset does not reflect that trend.
- This could be because:
  - The dataset was balanced across protocols for training purposes
  - Attackers are simulating legitimate traffic patterns
  - Or protocols alone aren’t enough to indicate risk without deeper packet or behavioral context

Insights from Visualization

Key Insight:

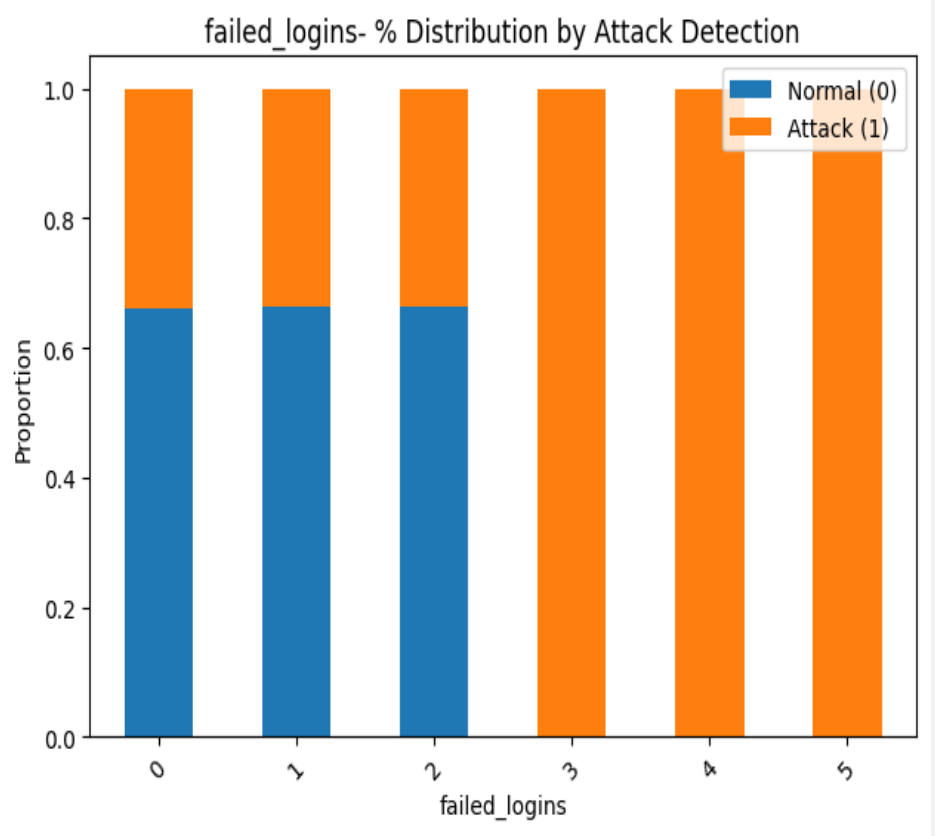
As the number of **failed login attempts increases**, the **likelihood of an attack increases significantly**.

For failed_logins = 0 to 2: Majority of sessions are **benign (~65%)**

For failed_logins ≥ 3: Sessions are **overwhelmingly attacks (~100%)**

Cybersecurity Interpretation:

- **Multiple failed login attempts** often indicate **brute-force** or **credential stuffing** attacks.
- Most legitimate users log in successfully or fail only once or twice.
- Attackers may try **many credentials rapidly**, triggering a high number of failures.



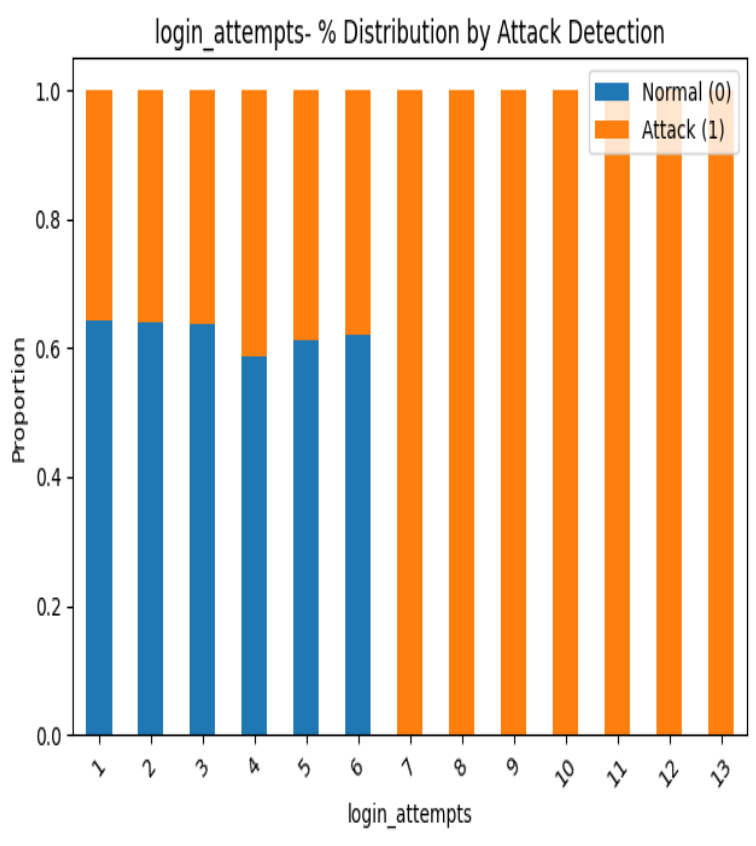
Insights from Visualization

Observations:

- For login_attempts from 1 to 6:
  - The majority of sessions are benign, though risk slowly increases.
- At 7 or more login attempts
  - The vast majority of sessions are attacks —
  - reaching near 100% at higher levels.
- Legitimate users rarely try to log in more than once or twice before succeeding or giving up.

Multiple attempts are indicative of:

- Credential brute-forcing
- Dictionary attacks
- Botnet-based mass login attempts



Over-all EDA Takeaways

Feature	Predictive Value	Key Insight
failed_logins	Very strong	3+ failures = likely attack
login_attempts	Very strong	7+ attempts = likely brute-force
browser_type	Strong	"Unknown" = suspicious agent
ip_reputation_score	Moderate	Wide variance = mixed-risk IPs
encryption_used	Weak	No clear attack preference
protocol_type	Weak	Evenly distributed among classes

Data Preprocessing Summary

Step	Description
Standardization	Normalize numeric features (mean = 0, std = 1) using StandardScaler
Ordinal Encoding	Encode ordered categories (e.g., severity levels) into integers
One-Hot Encoding	Convert nominal categories (e.g., protocol type) into binary columns
Feature & Target Definition	X = features (numeric + encoded), y = target (e.g., is_attack)
Stratified Train-Test Split	Split data while preserving class distribution (e.g., attack vs normal)

Model Building

Logistic Regression

- It's fast, interpretable, and gives clear insight into which features are driving predictions (great for SOCs and SHAP)
- **Type:** Linear model
- **Used for:** Establishing a strong baseline.
- **Strengths:**
  - Simple, interpretable
  - Useful for probabilistic predictions
- **Key Hyperparameters Tuned:**
  - C: Inverse of regularization strength (higher C = less regularization
  - Solver: Optimization algorithm (liblinear used for small, sparse datasets)

Model Building

Random Forest

- **Type:** Ensemble of decision trees
- **Used for:** Handling nonlinear patterns and interactions between features

- **Strengths:**
  - Resistant to overfitting
  - Captures complex feature relationships
- **Key Hyperparameters Tuned:**
  - n_estimators: Number of trees
  - max_depth: Controls tree growth to prevent overfitting
- **Uses in cybersecurity:** Trees can model complex conditions like “if login failures are high and IP risk is high”, making it ideal for behavioral rule learning

Model Building

XGBoost (Extreme Gradient Boosting)

- **Type:** Gradient boosting model (additive ensemble)
- **Used for:** High-performance learning in imbalanced classification
  - **Strengths:**
    - Excellent accuracy, especially on structured/tabular data
    - Handles missing data, robust to outliers
  - **Key Hyperparameters Tuned:**
    - n_estimators: Number of boosting rounds
    - max_depth: Depth of each tree
    - learning_rate: Step size in boosting (controls how fast model learns)
  - **Uses in cybersecurity:** It's one of the most powerful models for attack prediction, especially when paired with SHAP for interpretability.

Model Evaluation: Cross-Validation & Hyperparameter Tuning

Concept	What It Means	How We Used It
Cross-Validation	Repeatedly splits data into training and validation folds to evaluate model	Used **5-Fold Cross-Validation** to ensure stable F1-score estimates
Why It Matters	Prevents overfitting, gives more robust performance estimation	Ensured model generalizes well to unseen data
Hyperparameter Tuning	Systematic search for best model settings (e.g., depth, learning rate)	Used **GridSearchCV / RandomizedSearchCV** to find optimal model configurations
Tuned Models	Random Forest, XGBoost	Tuned parameters like max_depth, n_estimators, learning_rate, etc.
Selection Metric	Evaluation score used for choosing best model	Used **F1-Score** to account for class imbalance in attack detection

Model Evalution

All three models have excellent precision (very few false positives).

Random Forest has the highest F1-score, indicating the best balance between detecting threats and avoiding noise.

Metric	Logistic Reg.	Random Forest	XGBoost
Precision (1)	0.983	1.000	1.000
Recall (1)	0.742	0.743	0.741
F1-Score (1)	0.846	**0.853**	0.851

Model Selection

Using the Confusion Matrix to Cross-Check:

- **False negatives** (missed attacks)
- **False positives** (normal sessions wrongly flagged)

In your matrices:

- All models catch 742–743 attacks out of 853 → similar recall
- Logistic Regression has **slightly more false positives** than the others.
- **Decision Based on Use Case: We Choose Random Forest**:
- Best **F1-score (0.853)**
- Perfect **precision (1.000)**
- Solid **recall (0.743)**
- Easy to interpret via SHAP or feature importance

If You Want…	Choose…
Best **overall detection balance**	**Random Forest**
Highest **interpretability**	Logistic Regression
Faster runtime & deployment	XGBoost

Model Building

Neural network

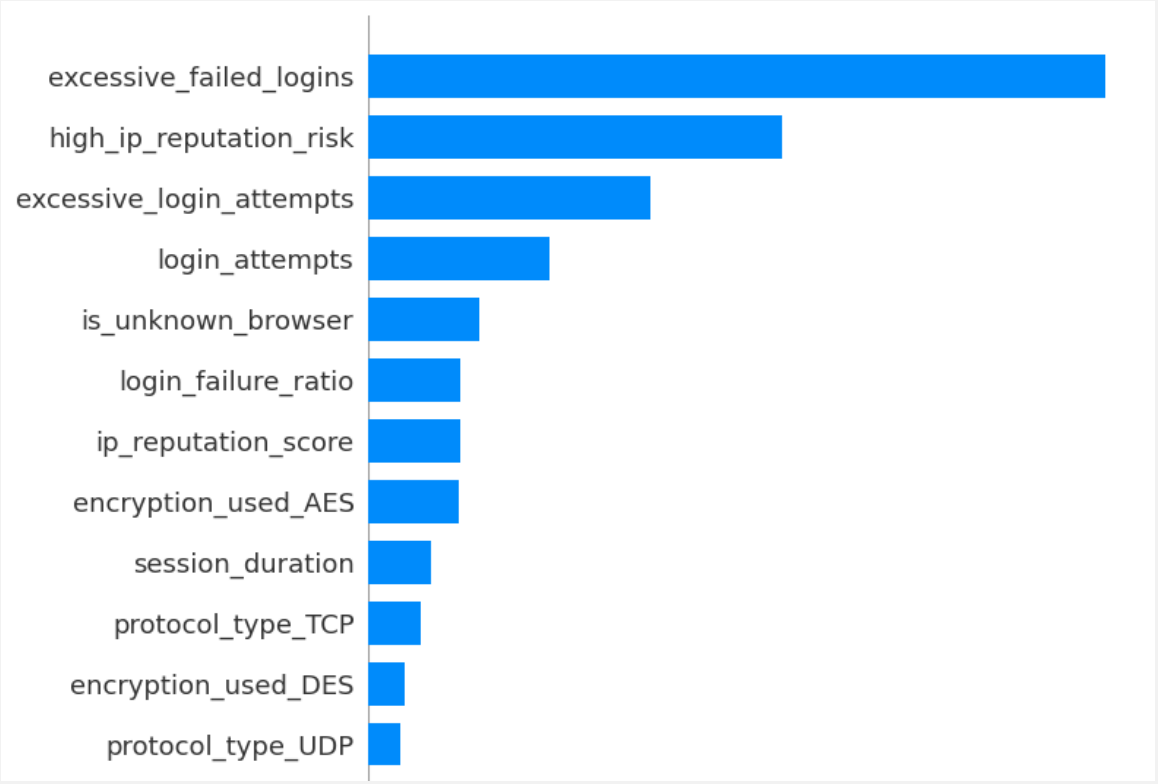
- A neural network was trained with varying epochs and batch sizes to evaluate performance.
- Hyperparameter grid search included:
  - -Epochs: 30, 60
  - -Batch Sizes: 32, 64, 100
- Metrics recorded: Accuracy, Precision, Recall, F1-Score.
- Best model performances:
  - - Deep Learning Precision: 0.978
  - - Classic ML Precision (Random Forest/XGBoost): 1.000
- Despite slightly lower precision, the DNN generalizes well.

Layer	Type	Size	Activation	Role
1	Dense	128	ReLU	Learns complex nonlinear patterns from input
2	Dense	64	ReLU	Refines learned interactions
3	Dense	32	ReLU	Adds more depth and abstraction
4	Output Layer	1	Sigmoid	Outputs probability of class 1 (attack)

SHAP Analysis

- SHAP was used to understand feature importance in predictions.
- Most influential feature: excessive_failed_logins
- Indicates model is learning relevant behaviors
- Three new variables were engineered:

  - excessive_failed_logins
  - login_failure_ratio
  - unusual_time_access



Network Modeling & Anomaly Detection

Introduction : In **cybersecurity**, network modeling helps us understand what "normal" traffic behavior looks like in an organization. Once a baseline of normal activity is established, **machine learning** is used to detect any unusual or suspicious patterns — these anomalies could represent **intrusions, malware activity, or insider threats**.

Perspective	Key Concepts
Cybersecurity	- Understand "normal" traffic behavior - Flag **deviations as threats** (e.g., intrusions, malware)
	- Detect **zero-day / unknown** attacks - Mimics SOC workflow: **monitor → flag → investigate**
Machine Learning	- Train on **unlabeled sessions flow** data - Use models like **Autoencoders, Isolation Forest,One class SVM**
	- Score based on **deviation from normal** - No need for attack labels → **generalizes well**

Methodology

Phase	Step	Details
🔧 **Phase 1: Network Modeling**	**Class Isolation**	Train only on sessions data without any label to learn baseline behavior
	Model Selection	Choose from unsupervised models: • Isolation Forest • One-Class SVM • Autoencoder
	Model Training	Fit the model to normal data to capture standard traffic patterns
🧪 **Phase 2: Anomaly Detection**	**Scoring New Data**	Pass unseen data (including attacks) through the trained model
	Anomaly Score	Calculate how far each session deviates from normal behavior
	Thresholding	Flag sessions as anomalies if score exceeds a predefined threshold,tto reduce FP
📊 **Phase 3: Evaluation**	**Label Comparison**	Compare model predictions with true labels (ground truth)
	Metrics Used	Use: • F1-score (for Class 1 - Attack) • Precision • Recall • Accuracy
	Cross-validation	Apply k-fold CV to evaluate model robustness and avoid overfitting

Model Building

MODEL	OBJECTIVE	HOW IT WORKS (STEP-BY-STEP)	KEY ASSUMPTION / INSIGHT
1 **Isolation Forest**	Identify anomalies by isolating observations through random decision trees	- Build multiple random trees by selecting random features & split values - Recursively partition each data point until isolated - Record path length for each point - Normal points → longer paths - Anomalies → shorter paths - Compute average path length as anomaly score	Anomalies are few and easily separable — they get isolated quickly in fewer splits.
2 **One-Class SVM**	Enclose all normal data in a tight boundary — flag outliers	- Train only on normal data points - Use RBF kernel to project into higher dimensions - Fit a hypersphere/hyperplane enclosing most of the normal data - For new data: • Inside boundary → normal • Outside → anomaly - Output: binary label or anomaly score	Assumes normal data is densely clustered; outliers lie outside the learned boundary.
3 **Autoencoder**	Reconstruct normal inputs; detect anomalies via reconstruction error	- Build an encoder–decoder neural network - Encoder compresses input - Decoder reconstructs input - Train only on normal data - For new data: • Reconstruct input • Compute reconstruction error (e.g., MSE) - If error > threshold → flag as anomaly	High reconstruction error = data point differs significantly from learned normal patterns.

Model Comparison

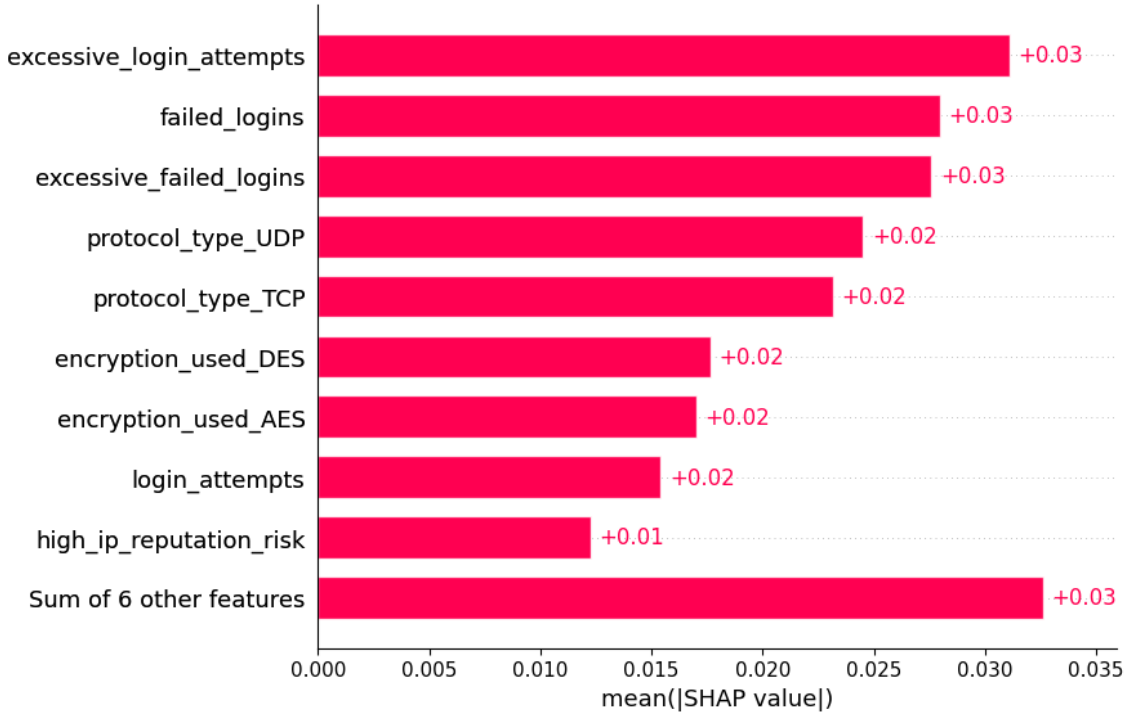
Model	How It Works	Strengths	Weaknesses
Isolation Forest	Random trees isolate points quickly	Fast, works well for clear outliers	Poor at subtle anomaly detection
One-Class SVM	Finds boundary around normal cluster	Precise, kernel flexibility	Slower, sensitive to scaling
Autoencoder	Learns to reconstruct normal behavior	Very accurate, flexible	Needs more training time

Model Evaluation – Class 1 (Attack)

Model	Precision (1)	Recall (1)	F1-Score (1)
Isolation Forest	0.557	0.311	0.399
One-Class SVM	0.761	0.787	0.774
Autoencoder	**0.865**	**0.792**	**0.827**

SHAP ANALYSIS

The SHAP plot explains which **input features** contributed **most** to **high reconstruction error** — which is how the model detects **anomalies**.



Component	Explanation
Autoencoder	Trained only on normal sessions to **reconstruct input** features accurately
Reconstruction Error	Measures how **different** the reconstructed output is from the actual input
High Error	Means the session is **unusual or anomalous** — likely an attack
SHAP	Attributes this error **back to features** that confused the model most

Model Evaluation – Class 1 (Attack)

Aspect	Supervised Learning	Unsupervised Learning (Anomaly Detection)
Goal	Learn to predict a known **label** (e.g., "attack" or "normal")	Find **patterns** and flag **unusual or rare events** (without labeled guidance)
Data Requirement	Requires a dataset with **labeled outcomes** for each input	Uses **unlabeled data**, typically trained only on **normal behavior**
Typical Models	Logistic Regression, Random Forest, XGBoost, Neural Networks	Isolation Forest, One-Class SVM, Autoencoder
Output	Predicts a class (e.g., "attack") or probability	Flags anomalies based on **deviation from normal patterns**
Use Case	When labeled data is available, and attacks are well-known and consistent	When attack types are **unknown**, rare, or **evolving**
Interpretability	Explains **why** the model predicts a specific label	Explains **why** the data point looks **abnormal or unexpected**
Evaluation	Based on classification metrics: **Precision, Recall, F1-score**	Based on **reconstruction error** or **anomaly score thresholds**
Strengths	High accuracy on known attack types	Can detect **zero-day, novel, or subtle threats**
Limitations	Can’t detect unseen threats or label drift	May generate false positives on edge cases or outliers

Conclusion Statement

Our hybrid intrusion detection approach — combining supervised classification with unsupervised anomaly detection — provides a scalable, intelligent defense system that detects both known and novel cyber threats. By embedding behavioral intelligence into our features and validating decisions with SHAP, we ensure not just high detection performance, but also transparency and real-world interpretability. This approach mirrors how modern SOCs operate and is deployable in real enterprise environments.

Key Takeaways & Insights

Area	Insight
Behavioral Features	Features engineered from login and IP behavior were top predictors
Supervised Learning	Random Forest had highest detection performance for labeled attacks
Unsupervised Learning	Autoencoder best captured anomalies among unlabeled sessions
Explainability	SHAP revealed that models learned human-reasonable attack signals
Realism	Training anomaly models only on normal data mimics SOC operations
Full Coverage	Combining both learning types ensures better detection of both known and emerging threats
Consistency	Using the same feature pipeline helped unify model logic and comparability

Future Scope and Insights

Recommended Multi-Layered Detection Approach

- Use Autoencoder as the first-stage anomaly filter, flagging unknown behaviors for review.
- Follow with DNN or Random Forest for confirmation, reducing false positives while ensuring no threats are missed.
- Leverage Logistic Regression for insights and audits into what features are influencing detection — useful in fine-tuning SIEM rules or explaining alerts to leadership.

Model-Guided Honeypots: Learn the Threat

- The models respond well to traffic bursts, failed logins, and unencrypted sessions.
- Use these findings to design smarter honeypots that simulate high-risk behaviors (e.g., unencrypted admin pages with high login activity).
- Feed traffic into your models to measure how quickly they detect and react to synthetic intrusions — turning your model into a real-time threat evaluation tool.