Cybersecurity Intrusion Detection System (IDS).

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 intrusions
- Unsupervised learning (without labels) to detect novel, unknown threats via anomaly detection

We apply these techniques on the same dataset to simulate real-world operational challenges where labeled data is limited and threats constantly evolve.

What is the problem?

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

Our approach

We mirror the operational realities of a Security Operations Center (SOC):

Use labeled data to train classifiers that mimic analyst triage for known attacks.

- The model learns from historical data labeled by human analysts (e.g., which sessions were confirmed intrusions vs. harmless).
- It automatically predicts whether new sessions are likely to be intrusions.
- This reduces the burden on human analysts by handling routine classification.

Use unlabeled data to flag previously unseen attack types using anomaly detection.

- Use unlabeled data to detect anomalous sessions that deviate from typical network behavior.
 The model learns what “normal” activity looks like by analyzing patterns across all sessions — without needing predefined attack labels.
- It then flags unusual or outlier sessions that may represent novel, stealthy, or zero-day threats.
- This approach helps detect unknown attacks and provides early warning signals, augmenting analyst capabilities in identifying emerging risks.

By leveraging both types of learning:

- Supervised learning enables the system to mimic analyst decision-making and accurately classify known intrusions based on historical labels.

- Anomaly detection empowers the system to identify novel or evolving threats by flagging unusual behaviors in unlabeled data.

Together, they create a comprehensive intrusion detection framework that balances precision with adaptability, ensuring both coverage of known threats and detection of emerging risks in real-time SOC environments.

EDA :

Data Overview

We worked with a **session-level cybersecurity dataset** containing metadata from **9,537 network sessions**. Each row represents one session, with fields that capture various characteristics relevant to network security, user behavior, and potential intrusion indicators.

📁 Dataset Dimensions

- **Total Records**: 9,537 sessions
- **Total Features**: 11 columns

🧾 Columns Description

Feature	Type	Description
session_id	Object	Unique identifier for each session (not used for modeling)
network_packet_size	Integer	Total size (in bytes) of packets exchanged during the session
protocol_type	Object	Type of network protocol used (e.g., TCP, UDP)
login_attempts	Integer	Number of login attempts made in the session
session_duration	Float	Duration of the session (in seconds)
encryption_used	Object	Type of encryption applied (e.g., AES, DES, or None)
ip_reputation_score	Float	Trustworthiness of the IP address (higher score indicates a safer connection)
failed_logins	Integer	Count of failed login attempts
browser_type	Object	Browser or user agent used during the session
unusual_time_access	Integer	Flag (0 or 1) indicating access during unusual hours

attack_detected	Integer	Target variable: 1 if an intrusion was detected, 0 otherwise

Handling Missing values :

During initial data inspection, we found that **only one feature contained missing values**:

- **encryption_used** had **1,966 missing entries** out of 9,537 records.

Why it matters:

In a cybersecurity context, encryption status is a **key security signal**. Sessions without encryption could be:

- Indicative of insecure communication (e.g., plain-text protocols)
- More vulnerable to interception or attacks
- A meaningful predictor of malicious behavior

Strategy Applied:

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**

Instead, we:

- **Replaced missing values with 'None'**
- This explicitly captures the idea of **unencrypted or unknown encryption sessions**

```
df['encryption_used'] = df['encryption_used'].fillna('None')
```

Updated Value Counts:

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

Data cleaning:

```
df = df.drop(columns=['session_id'])
```

We have dropped `session_id` because:

- It’s a non-informative, unique identifier
- It would hurt model generalization if left in
- It’s not useful for making predictions about whether a session is malicious

Oversampling :

Oversampling is a technique used to **artificially increase the number of minority class examples** in a dataset — typically in binary classification problems with **class imbalance**.

In your dataset:

- `attack_detected = 1` → Intrusion
- `attack_detected = 0` → Normal session

In real-world scenarios, **attacks are rare** (usually <1% of traffic). But here, attacks are **~45%** of the dataset.

That suggests the dataset was **intentionally oversampled**, meaning:

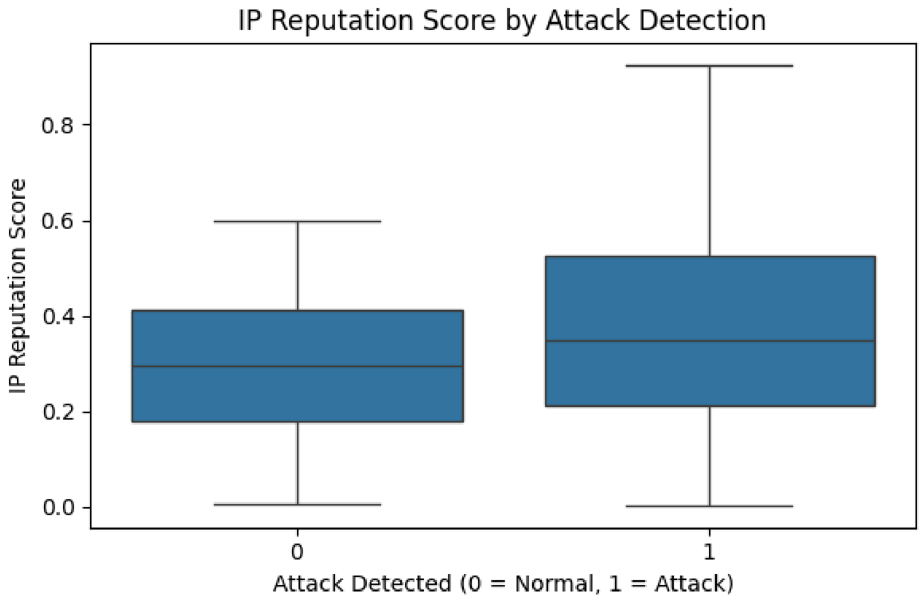
> More attack sessions were added (replicated or sourced) to balance the dataset.

Why Is Oversampling Needed in Cybersecurity ML?

Because **attack data is rare**, but **critically important**:

- 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 :



IP Reputation Score vs. Attack Detection

What This Shows

This boxplot compares the **IP Reputation Score** for:

- 0 = **Normal sessions**
- 1 = **Attack sessions**

Observations:

- **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.

Interpretation:

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

This includes:

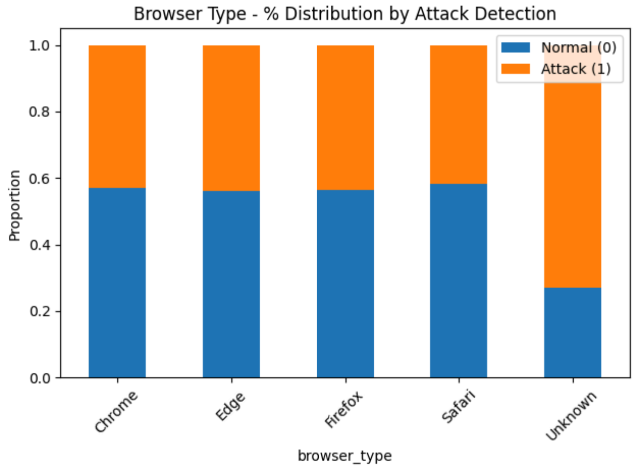
- **Low-reputation IPs** (e.g., previously flagged for spam, malware, or botnet behavior)
- **Some high-reputation IPs**, possibly due to:
    - IP spoofing
    - Compromised legitimate machines
    - Attempts to blend in with normal traffic

Meanwhile, benign users typically access from **mid-tier or consistently safe IPs**.

Cybersecurity Insight:

- IP reputation is a **useful feature** for detecting malicious behavior, but:
    - It should be **used with caution**, as some attack traffic may come from **temporarily clean or misclassified IPs**.
    - Models must **combine this feature with others** (e.g., login attempts, session duration) for reliable predictions.

Browser Type Distribution by Attack Detection



Key Insight:

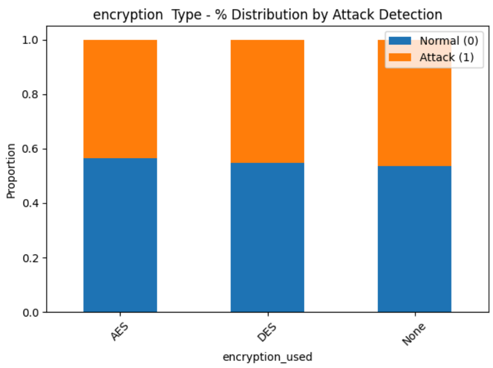
Sessions using an **"Unknown" browser type** are disproportionately malicious.

- **~70%** of sessions with `browser_type = Unknown` were flagged as **attacks**
- In contrast, common browsers (Chrome, Firefox, Safari, etc.) have ~55–60% benign sessions
- This suggests the "Unknown" browser is a **strong signal of suspicious or automated traffic**, possibly from:
  - Bots
  - Scripts
  - Malicious crawlers
  - Obfuscated agents

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.

Encryption Type vs. Attack Detection



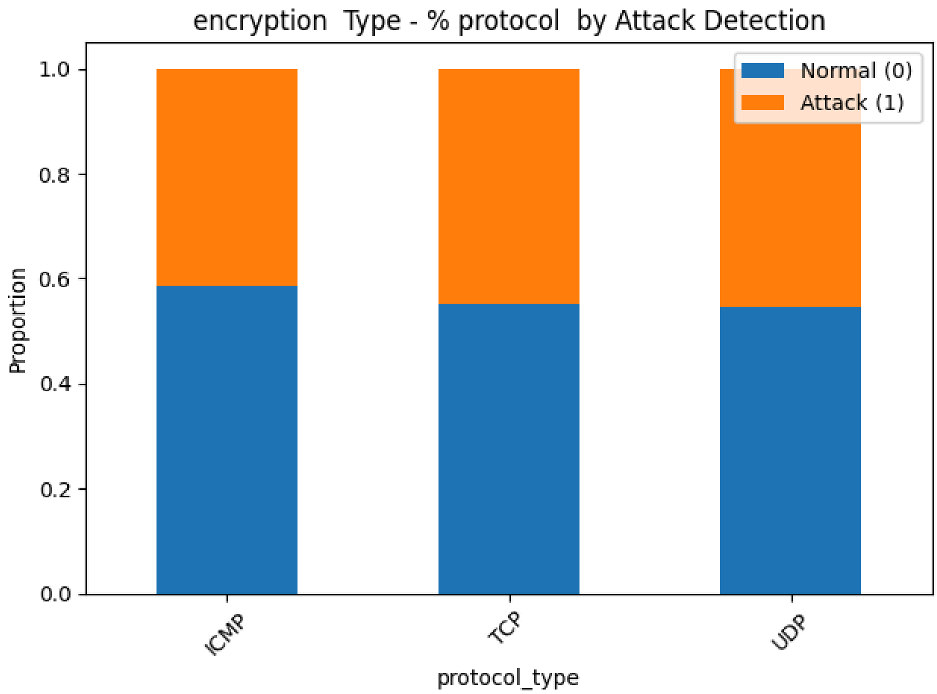
Interpretation:

- 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."

Protocol Type vs. Attack Detection



Interpretation:

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

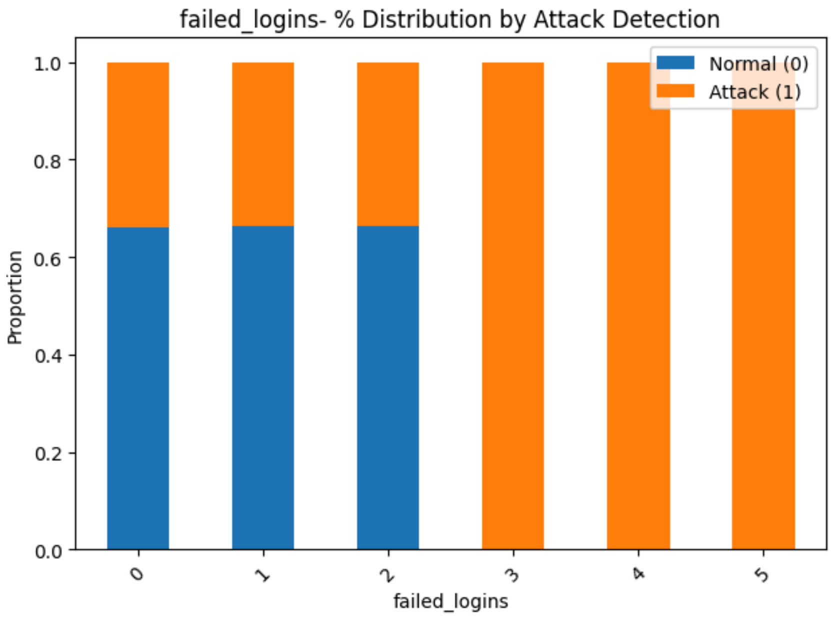
Cybersecurity Insight:

While certain protocols are commonly abused in real-world attacks (e.g., UDP for DDoS, ICMP for scanning), that trend is not reflected in this dataset.

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

Failed Logins vs. Attack Detection



Key Insight:

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

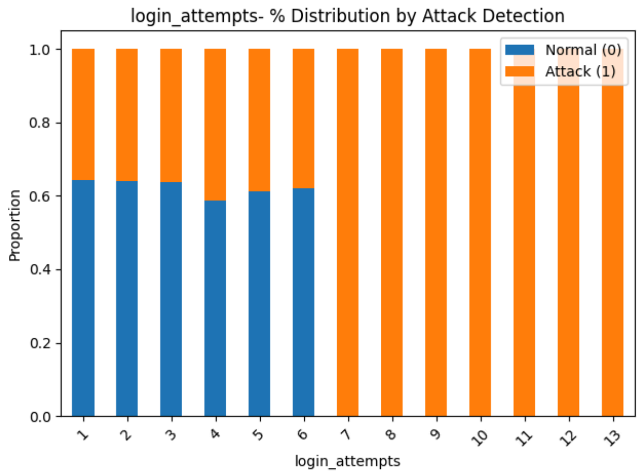
- For failed_logins = 0 to 2:
  - Majority of sessions are **benign (~65%)**
- For failed_logins ≥ 3:
  - Sessions are **overwhelmingly attacks (~100%)**

Cybersecurity Interpretation:

This makes sense operationally:

- **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.

Login Attempts vs. Attack Detection



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**.

🧠 Real-World Cybersecurity Context

This pattern aligns strongly with known attacker behaviors:

🚨 Why Many Login Attempts Are Risky:

- 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**
  - **Misconfigured scripts probing credentials**

📋 SOC Rule Examples:

- "Alert if login_attempts > 5 for any user within 5 minutes"
- "Block IP if >10 login attempts observed in a session"

So, this behavior reflects exactly what a **Security Operations Center (SOC)** would monitor and act on in production.

Overall 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

Feature Engineering

Feature Name	Code Snippet	Purpose	Why It's Good
excessive_login_attempts	df['excessive_login_attempts'] = (df['login_attempts'] >= 7).astype(int)	Flag brute-force attempts with 7 or more login attempts	Matches EDA; simplifies brute-force detection; improves model interpretability
high_ip_reputation_risk	df['high_ip_reputation_risk'] = (df['ip_reputation_score'] >= 0.6).astype(int)	Flag sessions from IPs with high-risk reputation scores	Tunable risk threshold; helps highlight IP-based threats; works well with tree-based models
login_failure_ratio	df['login_failure_ratio'] = df['failed_logins'] / df['login_attempts'].replace(0, 1)	Capture % of failed login attempts regardless of count	Captures subtle attacks; adds nuance beyond absolute numbers; effective for low-volume threats
excessive_failed_logins	df['excessive_failed_logins'] = (df['failed_logins'] >= 3).astype(int)	Flag sessions with 3 or more failed login attempts	Mirrors SOC detection rules; strong indicator of credential abuse

is_unknown_browser	df['is_unknown_browser'] = (df['browser_type'] == 'Unknown').astype(int) df = df.drop('browser_type', axis=1)	Identify automated tools or crawlers using non-standard user agents	Flags suspicious/masked clients; simplifies noisy categorical variable into a strong binary signal

Data Preprocessing Summary

🔹 1. Standardization of Numeric Features

```
numeric_cols = ['network_packet_size', 'session_duration', 'ip_reputation_score', 'login_failure_ratio']
scaler = StandardScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

✅ Why It's Good:

- Ensures numeric features are on the same scale (mean = 0, std = 1)
- Important for logistic regression, XGBoost, and distance-based models
- login_failure_ratio is especially sensitive and benefits from normalization

---

🔹 2. Ordinal Encoding Setup for Categorical Features

```
protocol_order = CategoricalDtype(categories=['ICMP', 'TCP', 'UDP'], ordered=True)
df['protocol_type'] = df['protocol_type'].astype(protocol_order)

encryption_order = CategoricalDtype(categories=['None', 'AES', 'DES'], ordered=True)
df['encryption_used'] = df['encryption_used'].astype(encryption_order)
```

✅ Why It's Good:

- Explicitly sets the base category for modeling (ICMP and None)
- This makes interpretation of one-hot encoded features more meaningful
- Reflects domain logic — None as base for encryption is security-aligned

---

🔹 3. One-Hot Encoding of Categorical Variables

```
df = pd.get_dummies(df, columns=['protocol_type', 'encryption_used'], drop_first=True)
```

✅ Why It's Good:

- Converts categorical features into a format suitable for ML models
- drop_first=True avoids multicollinearity and creates a reference group
- Now protocol_type_TCP, protocol_type_UDP, encryption_used_AES, encryption_used_DES will represent comparison to base levels

---

🎯 Model Training Setup

🔹 4. Defining Features and Target

```
X = df.drop(columns=['attack_detected'])
y = df['attack_detected']
```

- X: All predictive features (both original and engineered)
- y: Binary target variable (0 = normal, 1 = attack)

🔹 5. Stratified Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=42, stratify=y
)
```

✅ Why It's Good:

- Stratification preserves class balance in both training and test sets
- Ensures fair evaluation metrics, especially in classification problems

Model building

Logistic Regression

- 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)

✅ Why it's useful in cybersecurity: It's fast, interpretable, and gives clear insight into which features are driving predictions (great for SOCs and SHAP).

2️⃣ 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

✅ Why it's useful 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.

3️⃣ 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)

✅ Why it's useful in cybersecurity: It's one of the most powerful models for attack prediction, especially when paired with SHAP for interpretability.

🔧 Hyperparameter Tuning with GridSearchCV

📌 What is GridSearchCV?

GridSearchCV is a method that:

- Exhaustively searches through a grid of hyperparameter combinations

- Evaluates each combination using cross-validation
- Selects the best combination based on a scoring metric (you used precision)

MODEL SELECTION :

Identify Critical Evaluation Metrics

For this binary classification task, especially in a SOC (Security Operations Center) context, we care about:

Metric	Meaning
Precision (1)	How many of the sessions predicted as attacks **were actually attacks**? (Avoid false alarms)
Recall (1)	How many of the actual attacks did we **successfully detect**? (Avoid missing threats)
F1-score (1)	Balance between precision and recall — good overall metric
Confusion Matrix	Shows the exact number of true/false positives/negatives

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

- 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**.

🔹 Step 5: Use the Confusion Matrix to Cross-Check

Look at the number of:

- **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

---

🔹 Step 6: Decide Based on Use Case

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

✅ **Choose Random Forest**:

- Best **F1-score (0.853)**
- Perfect **precision (1.000)**
- Solid **recall (0.743)**
- Easy to interpret via SHAP or feature importance

SHAP

Top Features Influencing Attack Prediction

Rank	Feature	SHAP Insight	Cybersecurity Interpretation
1️⃣	excessive_failed_logins	Largest contributor to predicting a session as an attack	🔐 Brute-force login behavior — repeated credential failures
2️⃣	high_ip_reputation_risk	Strong signal for attack prediction	🛑 IPs with known malicious history (e.g., spam, botnets, dark web)
3️⃣	excessive_login_attempts	Significant contribution toward attack label	🤖 Likely automated scripts or bots trying many credentials
4️⃣	login_attempts	Supports predictions where excessive attempts may not have crossed the flag threshold	📈 Adds gradient-level behavior context to binary excessive_... flags
5️⃣	is_unknown_browser	Important signal, contributes to attack label	👻 Browser obfuscation or bots — non-human or script-based traffic

6️⃣	login_failure_ratio	High ratios push prediction towards attack	🎯 Targeted brute-force or manual probing with low success rate

"The SHAP summary plot shows that our model relies most heavily on behavioral features that reflect brute-force login activity (excessive_failed_logins, login_attempts) and risk signals such as high IP reputation and unknown browsers. These align closely with how a real SOC triages threats, validating that our model is both accurate and interpretable."

Why This Is Powerful

- Confirms that feature engineering added strong, interpretable value.
- Demonstrates that the model is making decisions based on real, meaningful patterns, not random noise.
- Builds trust with stakeholders (e.g., cybersecurity teams, business users).

DEEP LEARNING

Model Architecture

```
model = Sequential([
   Dense(128, input_dim=X_train.shape[1], activation='relu'),
   Dense(64, activation='relu'),
   Dense(32, activation='relu'),
   Dense(1, activation='sigmoid')
])
```

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)

🔹 3. Loss Function & Optimizer

```
loss='binary_crossentropy', optimizer=Adam(0.001)
```

- Binary cross-entropy: Ideal for 2-class problems. Punishes incorrect predictions based on confidence.
- Adam: Adaptive optimizer — balances speed and convergence.

🔹 4. Class Imbalance Handling

class_weight = {'0': ..., '1': ...}

- Ensures the model doesn't bias toward majority class
- Helps model pay equal attention to both attacks and normal sessions

---

🔹 5. Training Strategy

You ran a grid search over:

- Epochs (30, 60): How many full passes through the training data
- Batch sizes (32, 64, 100): How many samples are processed before updating weights

You selected:

Epochs = 30, Batch Size = 32

based on the best F1-score for class 1 (attack).

⚙️ How the DNN Works Internally

1. Input Layer: Takes all your session features — numeric and encoded (e.g., failed_logins, ip_risk, etc.)
2. Forward Propagation:
    - Data is multiplied by weights, passed through ReLU
    - Each layer transforms inputs into a higher-level representation
3. Output Layer:
    - Produces a probability from 0 to 1 (how likely is this session an attack?)
4. Backpropagation:
    - Model compares prediction to ground truth
    - Updates weights via gradient descent to reduce prediction error
5. Repeat for all samples across all epochs

📊 Results Interpretation

🧾 Final Model Report (30 epochs, 32 batch size)

Metric	Class 0 (Normal)	Class 1 (Attack)
Precision	0.828	0.973
Recall	0.983	0.747
F1-score	0.899	0.845
Accuracy	-	0.877

Cybersecurity Interpretation

- ✅ Precision (1) = 0.973 → Nearly all predicted attacks are real → fewer false positives
- ✅ Recall (1) = 0.747 → Detects ~75% of actual attacks → strong threat coverage
- ✅ F1-score (1) = 0.845 → Balanced and effective overall detection

Network Modeling & Anomaly Detection

🔐 **Why Are We Doing This?**

In real-world cybersecurity:

- **Labeled attack data is rare or unavailable** (especially for **zero-day** or stealthy attacks)
- **New threats constantly emerge**, and supervised models trained on old labels can miss them
- SOC analysts can't manually label millions of sessions

So, we use **anomaly detection** to:

> Learn what "normal" looks like and flag anything that deviates as suspicious — possibly an attack.Learn what "normal" looks like and flag anything that deviates as suspicious — possibly an attack.

💡 **Why Are We Using the Same Dataset?**

We use the same cybersecurity session dataset because:

1. It has both **normal and attack sessions** — useful for **evaluating** performance even if we train **without labels**
2. It mimics a **real SOC workflow**, where we:

- **Train on normal data** only (unlabeled)
- **Test** anomaly detection models against actual known attacks for validation

This helps us simulate the real challenge:
🔎 **Detecting unknown intrusions without pre-labeled attacks.**

🚧 What Is Network Modeling in This Context?

You're modeling the **distribution of benign (normal) session traffic**:

- IPs, login attempts, packet size, browser types, reputation scores, etc.
- All this helps your models understand **what a "normal" session should look like**
- Any session **outside that normal profile = anomaly = possible intrusion**

🧪 The Anomaly Detection Models Used

###1️⃣ Isolation Forest

🔧 **How It Works:**

- Creates multiple random trees
- **Normal data** takes **more splits to isolate**
- **Anomalies** are easier to isolate (closer to root of tree)

📊 **Used For:**

- Quick outlier detection
- Works well for high-dimensional structured data

⚠️ **Limitation:**

- Assumes anomalies are **few and very different**, so not good at **catching subtle attacks**
- Gave **low recall (0.311)** — missed many real attacks

###2️⃣ One-Class SVM

🔧 **How It Works:**

- Tries to draw a **tight boundary** around all normal data points
- Anything **outside** this boundary is flagged as an anomaly

📊 **Used For:**

- High-dimensional problems
- When "normal" data is **dense and consistent**

✅ **Strength:**

- Better **precision and recall balance**
- Captures nonlinear structures via the **RBF kernel**

###3️⃣ Autoencoder (Neural Network)

🔧 **How It Works:**

- Learns to **reconstruct normal input** (like compression → decompression)
- Trained **only on normal sessions**
- If the reconstruction error is **high**, the session is **anomalous**

📊 **Used For:**

- Deep, flexible modeling of complex patterns in "normal" data
- Great when anomalies are **subtle or context-dependent**

✅ **Why It Works Well:**

- Custom thresholding (90th percentile error on normal)
- Captures hidden patterns in behavior (e.g., slight changes in login failures + timing)

📌 Summary Table

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

.

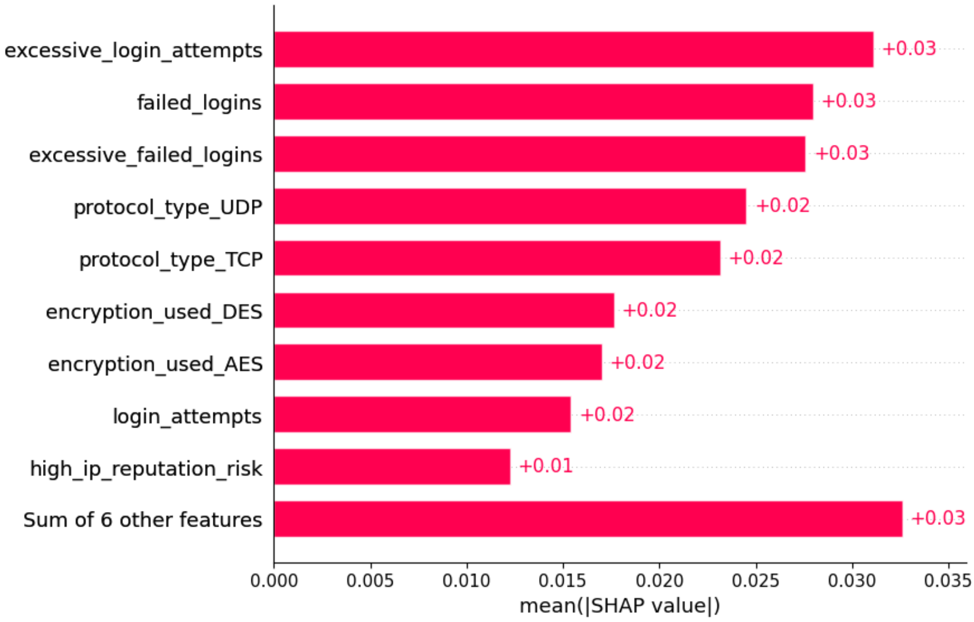
Model Evaluation – Class 1 (Attack)

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**

Model	Insight
Isolation Forest	Very low recall → misses most attacks, not reliable for your use case
One-Class SVM	Good recall and solid F1-score → works well in general scenarios
Autoencoder	Highest precision and best F1-score → best trade-off for SOC-style detection

SHAP

> **“SHAP interpretation of our autoencoder reveals that anomaly scores are primarily influenced by login behavior (excessive attempts, failures), protocol patterns, and encryption usage. This aligns with known intrusion behaviors and validates our feature engineering. The model effectively flags sessions that deviate from normal traffic based on real-world security signals.”**



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**