SSH Shell Attacks

ANDREA BOTTICELLA[*], Politecnico di Torino, Italy
ELIA INNOCENTI[*], Politecnico di Torino, Italy
RENATO MIGNONE[*], Politecnico di Torino, Italy
SIMONE ROMANO[*], Politecnico di Torino, Italy

This paper introduces a comprehensive machine learning approach to analyze SSH shell attack sessions, leveraging both supervised and unsupervised learning techniques. Using a dataset of 230,000 unique Unix shell attack sessions, the framework aims to classify attacker tactics based on the MITRE ATT&CK framework and uncover latent patterns through clustering. The key contributions of this work are:

- Development of a robust pre-processing pipeline to analyze temporal trends, extract numerical features, and evaluate intent distributions from large-scale SSH attack session data.
- Implementation of supervised classification models to accurately predict multiple attacker tactics, supported by hyperparameter tuning and feature engineering for enhanced performance.
- Application of unsupervised clustering techniques to uncover hidden patterns in attack behaviors, leveraging visualization tools and cluster analysis for fine-grained categorization.
- Exploration of advanced language models, such as BERT and Doc2Vec, for representation learning and fine-tuning to improve intent classification and session interpretation.

CCS Concepts: • **Computing methodologies** → **Supervised learning by classification**; **Unsupervised learning**; *Natural language processing*; *Machine learning*; *Machine learning approaches*; • **Security and privacy** → Intrusion detection systems.

Additional Key Words and Phrases: Machine learning, supervised learning, unsupervised learning, language models, text classification, clustering, intent classification, SSH shell attacks, security log analysis

Contents

1 INTRODUCTION

1.1 Motivation

Security logs play a crucial role in understanding and mitigating cyber attacks, particularly in the domain of network and system security. With the increasing sophistication of cyber threats, analyzing and interpreting security logs has become paramount for detecting, preventing, and responding to potential security breaches.

[*] The authors collaborated closely in developing this project.

Authors' Contact Information: Andrea Botticella, andrea.botticella@studenti.polito.it, Politecnico di Torino, Turin, Italy; Elia Innocenti, elia.innocenti@studenti.polito.it, Politecnico di Torino, Turin, Italy; Renato Mignone, renato.mignone@studenti.polito.it, Politecnico di Torino, Turin, Italy; Simone Romano, simone.romano@studenti.polito.it, Politecnico di Torino, Turin, Italy.

Publication date: January 2025.

Unix shell attacks, especially those executed through SSH, represent a significant vector for potential system compromises.

The complexity of security log analysis stems from several key challenges:

- Logs are often unstructured and contain ambiguous or malformed text.
- Manual parsing and interpretation of logs is time-consuming and error-prone.
- The sheer volume of log data makes comprehensive manual review impractical.

These challenges underscore the need for automated, intelligent approaches to log analysis that can efficiently extract meaningful insights and identify potential security threats.

1.2 Objective

The primary objective of this research is to develop and evaluate machine learning techniques for automatic analysis and classification of SSH shell attack logs. Specifically, we aim to:

- Automate the process of log analysis and intent classification.
- Provide security professionals with insights into attack strategies.

The significance of this research lies in its potential to enhance cybersecurity threat detection and response capabilities by transforming complex, unstructured log data into actionable intelligence.

2 BACKGROUND

2.1 Security Logs and Attack Analysis

In the context of SSH shell attacks, logs document the sequence of commands executed during a malicious session, enabling security researchers to analyze attacker behaviors, techniques, and potential system impacts. However, the manual analysis of these logs is challenging due to their volume, complexity, and often non-standard formatting.

2.2 MITRE ATT&CK Framework

The MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework provides a comprehensive knowledge base of adversary tactics and techniques observed in real-world cyber attacks. This framework serves as a standardized methodology for understanding and categorizing attack strategies.
For our research, we focus on seven key intents derived from the MITRE ATT&CK framework:

- **Persistence:** Techniques used by adversaries to maintain system access across restarts or credential changes.
- **Discovery:** Methods for gathering information about the target system and network environment.
- **Defense Evasion:** Strategies to avoid detection by security mechanisms.
- **Execution:** Techniques for running malicious code on target systems.
- **Impact:** Actions aimed at manipulating, interrupting, or destroying systems and data.
- **Other:** Less common tactics including Reconnaissance, Resource Development, Initial Access, etc.
- **Harmless:** Non-malicious code or actions.

2.3 Research Approach

Our research employs a multi-faceted approach to SSH shell attack log analysis:

- Explore and preprocess a large dataset of Unix shell attack sessions.
- Apply supervised learning techniques to classify attack tactics based on session characteristics.
- Utilize unsupervised learning methods to discover patterns and clusters in attack sessions.
- Investigate the potential of advanced language models in understanding and categorizing attack intents.

3 DATA EXPLORATION AND PRE-PROCESSING

3.1 Introduction

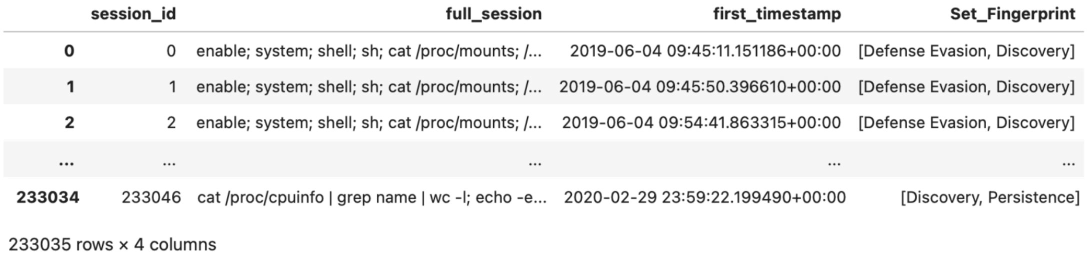
This section outlines the steps taken to explore and preprocess the dataset used in this study. The primary objective is to understand the data characteristics, identify patterns, and prepare the data for further analysis. We focus on temporal trends, intent distributions, and textual features.

3.2 Dataset Overview

The dataset consists of logs from SSH attacks. Each entry includes the following features:

- **session_id**: An integer representing the unique identifier for each session.
- **full_session**: A string containing the full sequence of commands executed during the attack session.
- **first_timestamp**: The timestamp indicating the start of the session.
- **Set_Fingerprint**: A set of strings (labels) describing the nature of the attack.

Below is an example of the dataset structure:



	session_id	full_session	first_timestamp	Set_Fingerprint
0	0	enable; system; shell; sh; cat /proc/mounts; /...	2019-06-04 09:45:11.151186+00:00	[Defense Evasion, Discovery]
1	1	enable; system; shell; sh; cat /proc/mounts; /...	2019-06-04 09:45:50.396610+00:00	[Defense Evasion, Discovery]
2	2	enable; system; shell; sh; cat /proc/mounts; /...	2019-06-04 09:54:41.863315+00:00	[Defense Evasion, Discovery]
...	...	...	...	...
233034	233046	cat /proc/cpuinfo | grep name | wc -l; echo -e...	2020-02-29 23:59:22.199490+00:00	[Discovery, Persistence]

233035 rows × 4 columns

Fig. 1. Dataset

3.3 Dataset Preparation and Feature Extraction

The dataset used in this study was provided as a Parquet file.

To prepare the dataset for analysis and ensure its quality, several preprocessing steps were undertaken, described in detail below:

1. **Decoding the full_session column**: The `full_session` column contained `90026` encoded shell scripts, making the raw data difficult to interpret. A decoding process was applied to these entries, converting them into a human-readable format.
2. **Formatting the first_timestamp column**: The `first_timestamp` column was checked for consistency and converted into a standard datetime format.
3. **Splitting the full_session column into lists of commands**: The `full_session` column contained entire attack sessions represented as single strings. Since we wanted each word / command to be a feature of our models, it was necessary to break it down into individual commands or keywords for more granular analysis. This was achieved through a multi-step splitting process using specific delimiters:
   - The semicolon (`;`) was used to separate distinct commands within a session.
   - The pipe (`|`) was used to divide concatenated commands or pipelines.
   - Whitespace (` `) was used to further split commands into individual words or arguments.

   This process allowed the identification of specific actions and parameters, facilitating a detailed analysis of the attack strategies.

(4) **Cleaning individual commands**: Each command or keyword extracted from the `full_session` column was processed to remove unnecessary or undesired elements. Regular expressions were used to strip unwanted symbols, qualifiers, and variable assignments.
(5) **Handling missing values**: To ensure the integrity and completeness of the dataset, a thorough check for missing values was performed across all columns.

```
enable ; system ; shell ; sh ; cat /proc/mounts ; /bin/busybox SAEMW ; cd /dev/
    shm ; cat .s || cp /bin/echo .s ; /bin/busybox SAEMW ; tftp ; wget ; /bin/
    busybox SAEMW ; dd bs=52 count=1 if=.s || cat .s || while read i ; do echo
    $i ; done < .s ; /bin/busybox SAEMW ; rm .s ; exit ;

[enable, system, shell, sh, cat, /proc/mounts, /bin/busybox, SAEMW, cd, /dev/shm
    , cat, .s, cp, /bin/echo, .s, /bin/busybox, SAEMW, tftp, wget , /bin/busybox
    , SAEMW, dd, bs, count, if, cat, .s, while, read, i, do, echo, i, done, .s,
    /bin/busybox, SAEMW, rm, .s, exit]
```

Listing 1. Dataset Processing

These preprocessing steps ensured that the dataset was well-structured, clean, standardized and ready for advanced analysis and machine learning tasks. By addressing issues such as encoding, formatting, and missing data, the preprocessing phase established a robust foundation for the study.

3.4 Temporal Analysis

The temporal analysis examines when the attacks were performed and the intent distribution over time.

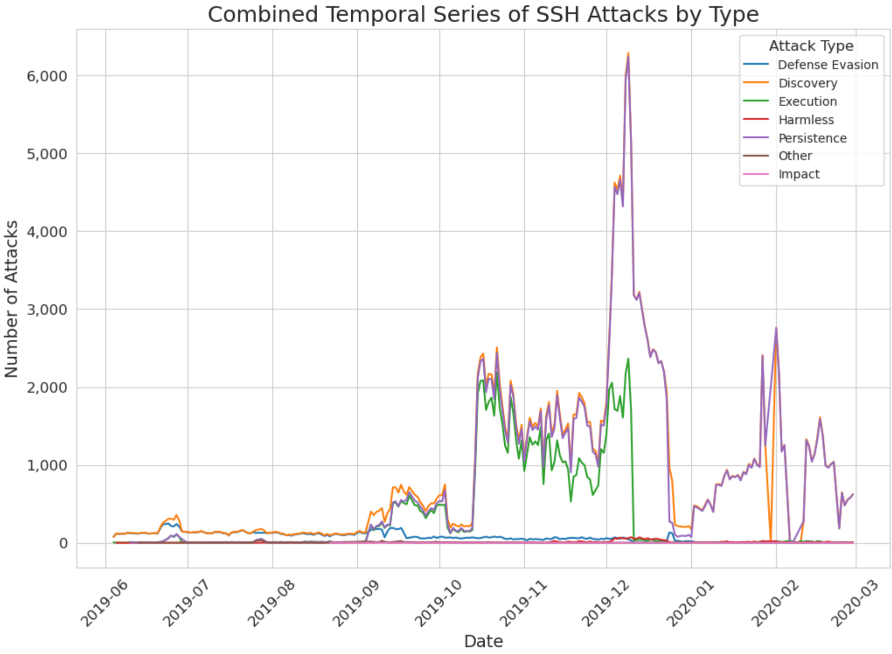


Fig. 2. Temporal Series of SSH Attacks

Our dataset spans from June 2019 to March 2020, revealing a notable concentration of attacks between mid-October 2019 and January 2020.

Further analysis was conducted on the distribution of attacks/sessions across various temporal dimensions (months, days of the week, etc.), but no significant patterns were identified.

3.5 Common Words Analysis

The analysis of common words within the attack sessions provides insights into frequently utilized commands and keywords. These elements are pivotal for understanding attacker behavior and informing model training for detecting attack intents.

To visualize and analyze the most frequent words, two approaches were used:

- **Word Cloud Visualization**: A word cloud (Figure 3a) was generated to represent the most common words in the dataset. Commands like `tmp`, `grep`, `var`, and `cat` dominate the word cloud, highlighting their frequent usage in attack sessions.
- **Top 20 Most Common Words Bar Plot**: A bar plot (Figure 3b) showcases the top 20 most frequent words, ordered by their frequency. This plot provides more quantitative detail, revealing the high occurrence of commands such as `tmp` (over 2 million occurrences) and `var` (nearly 2 million occurrences). Other significant commands include `grep`, `cat`, `echo`, and `cp`, each surpassing 1 million occurrences. These commands are commonly associated with file manipulation, string searching, and process information, which are typical in malicious activities.

The combination of these visualizations supports the identification of commonly used commands and their potential roles in attack sessions.

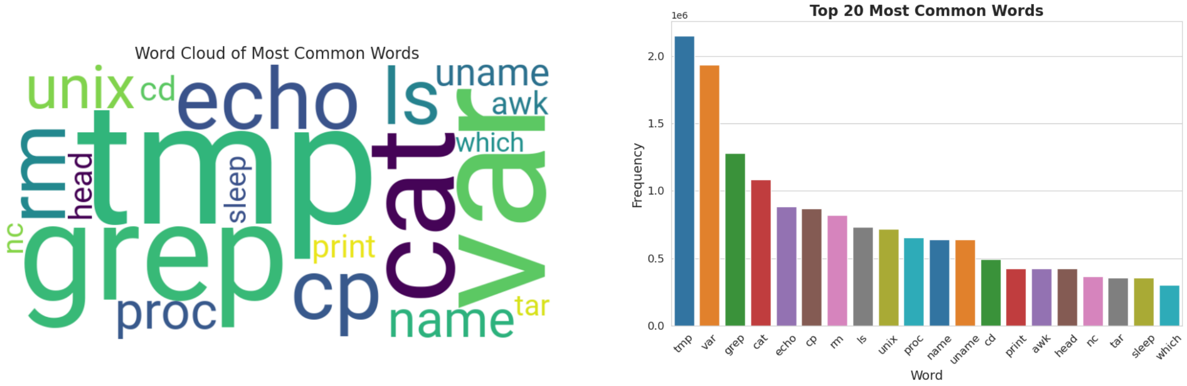


Fig. 3. (a) Word Cloud of Most Common Words and (b) Top 20 Most Common Words

3.6 Intent Analysis

This section presents an analysis of different intents identified in attack sessions and their relationships. Our analysis reveals distinct patterns in both the frequency of individual intents and their co-occurrence relationships, providing valuable insights into attacker behavior patterns.

Figure 4a illustrates the frequency distribution of different intents in our dataset. The analysis reveals that Discovery and Persistence are the most prevalent intents, with approximately 200,000 occurrences each. This suggests that attackers primarily focus on reconnaissance activities and establishing long-term presence in the targeted systems. Execution intent appears as the third most common, with roughly 100,000 occurrences, indicating

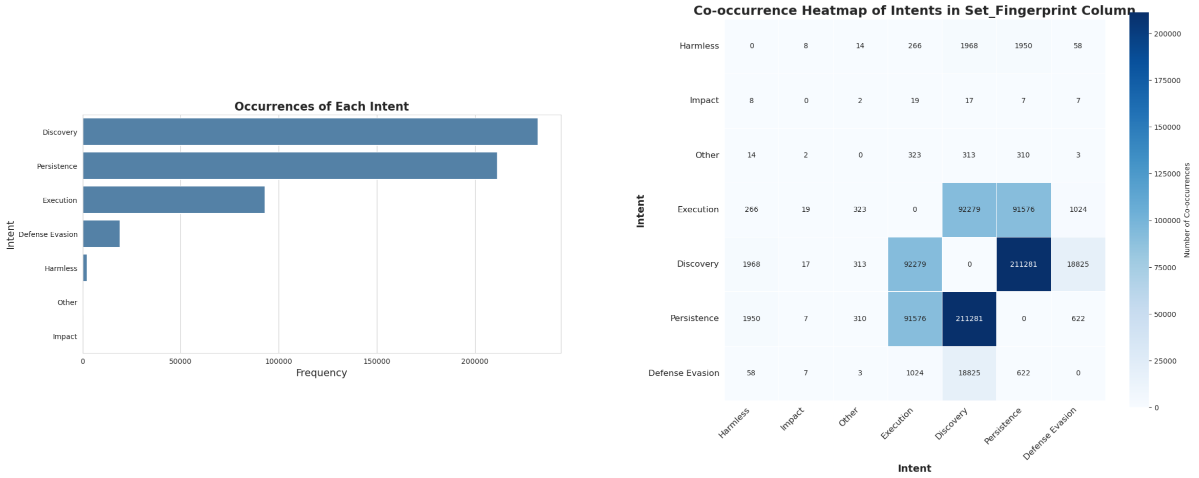


Fig. 4. (a) Occurrences of each Intent and (b) Co-Occurrence Heatmap of Intents

a significant number of attempts to run malicious code. Defense Evasion shows notably fewer occurrences (around 20,000), while Harmless, Other, and Impact intents are relatively rare in the dataset.

The co-occurrence heatmap (Figure 4b) reveals significant patterns in how different intents interact within attack sessions. Several key observations emerge:

- The strongest co-occurrence relationship exists between Discovery and Persistence (211,281 co-occurrences), suggesting that attackers often combine reconnaissance activities with attempts to maintain system access.
- Execution shows strong correlations with both Discovery (92,279) and Persistence (91,576), indicating that malicious code execution frequently accompanies both system discovery and persistence establishment attempts.
- Defense Evasion exhibits moderate co-occurrence with Discovery (18,825) and minimal correlation with other intents, suggesting that evasion techniques are primarily employed during reconnaissance phases.
- Harmless, Impact, and Other intents show minimal co-occurrence with other categories (with values mostly under 20 co-occurrences). This isolation pattern is partially explained by their low frequency in the dataset: Harmless with 2,206 occurrences, Impact with only 27 occurrences, and Other with 327 occurrences. These numbers represent a small fraction of the total recorded intents, naturally limiting their potential for co-occurrence with other intent types.

These findings highlight common attack patterns where adversaries typically begin with discovery operations, followed by persistence establishment and execution of malicious code. This understanding can inform the development of detection strategies and defensive measures, particularly focusing on the most common intent combinations.

3.7 Session Analysis

The session analysis aims to understand the structural characteristics of the attack sessions in the dataset. This is accomplished through two visualizations: the Empirical Cumulative Distribution Function (ECDF) plots and the distribution of the number of words per session.

The ECDF plots in Figure 5a provide insights into the distribution of the number of characters and words across all sessions. The Empirical Cumulative Distribution Function (ECDF) is a statistical tool that represents the proportion of observations below a certain value, offering an intuitive way to understand data distributions.

- The ECDF for characters indicates that the majority of sessions have fewer than 10^3 characters, with a sharp increase between 10^2 and 10^3 characters. This suggests a high concentration of relatively short sessions.
- The ECDF for words shows a similar trend, with most sessions containing fewer than 10^2 words. The distribution reflects the concise nature of many attack sessions, potentially focusing on executing a limited number of commands.

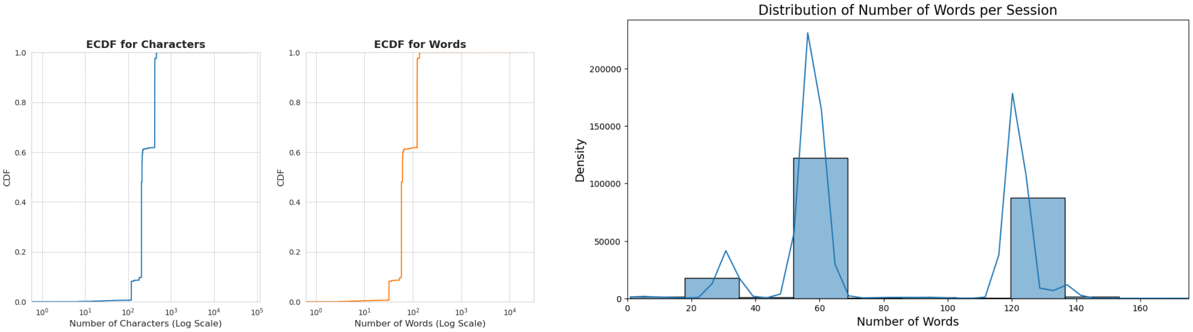


Fig. 5. (a) ECDF for Characters and Words in Sessions and (b) Distribution of the Number of Words per Session

Figure 5b illustrates the density distribution of the number of words per session. Key observations include:

- The distribution is bimodal, with two distinct peaks. This likely reflects different categories of sessions, such as exploratory attacks with a larger number of commands versus simpler, targeted attacks with fewer commands.
- The majority of sessions have fewer than 100 words, reinforcing the compactness observed in the ECDF plots.
- Outliers with higher word counts likely represent complex sessions, potentially involving multiple stages or more sophisticated attack strategies.

The relevance of these plots lies in their ability to provide a foundational understanding of session structure, which is critical for feature engineering and model development. Shorter sessions may correspond to quick, automated attacks, while longer sessions might represent manual or multi-step intrusions. These insights can inform the design of models by emphasizing features tailored to the varying lengths and complexities of sessions, ultimately improving detection accuracy.

3.8 Text Representation

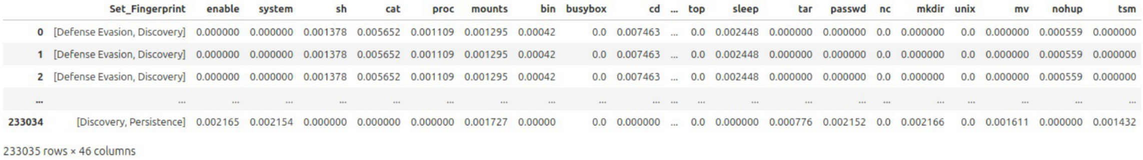
To enable further analysis of the dataset, the initial textual data was transformed into numerical representations. This step is essential for applying machine learning techniques, as numerical formats are required for model training and evaluation. Two widely used text representation techniques were employed:

- **Bag of Words (BoW)**: This technique represents each session as a vector where each dimension corresponds to a specific word, and the value represents the frequency of that word in the session. While simple and interpretable, BoW does not capture the importance or uniqueness of words across the dataset.
- **Term Frequency-Inverse Document Frequency (TF-IDF)**: This method extends BoW by weighting the word frequencies based on their inverse document frequency, thereby emphasizing words that are

important within a session but less frequent across all sessions. This approach helps reduce the impact of common but non-informative words.

Figure 6 and Figure 7 illustrate the transformed datasets using the BoW and TF-IDF techniques, respectively. The TF-IDF representation was selected for further analysis due to its ability to capture more meaningful patterns in the data by accounting for word relevance.

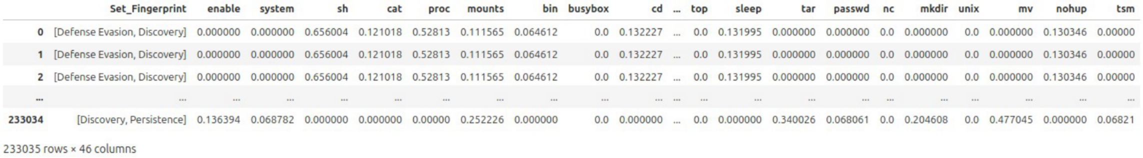
The use of the TF-IDF representation ensures that subsequent analyses focus on the most distinctive features of each session, enhancing the effectiveness of machine learning models in identifying patterns and predicting intents.



	Set_Fingerprint	enable	system	sh	cat	proc	mounts	bin	busybox	cd	...	top	sleep	tar	passwd	nc	mkdir	unix	mv	nohup	tsm
0	[Defense Evasion, Discovery]	0.000000	0.000000	0.001378	0.005652	0.001109	0.001295	0.00042	0.0	0.007463	...	0.0	0.002448	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.000559	0.000000
1	[Defense Evasion, Discovery]	0.000000	0.000000	0.001378	0.005652	0.001109	0.001295	0.00042	0.0	0.007463	...	0.0	0.002448	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.000559	0.000000
2	[Defense Evasion, Discovery]	0.000000	0.000000	0.001378	0.005652	0.001109	0.001295	0.00042	0.0	0.007463	...	0.0	0.002448	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.000559	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
233034	[Discovery, Persistence]	0.002165	0.002154	0.000000	0.000000	0.000000	0.001727	0.00000	0.0	0.000000	...	0.0	0.000000	0.000776	0.002152	0.0	0.002166	0.0	0.001611	0.000000	0.001432

233035 rows × 46 columns

Fig. 6. Bag-of-Words



	Set_Fingerprint	enable	system	sh	cat	proc	mounts	bin	busybox	cd	...	top	sleep	tar	passwd	nc	mkdir	unix	mv	nohup	tsm
0	[Defense Evasion, Discovery]	0.000000	0.000000	0.656004	0.121018	0.52813	0.111565	0.064612	0.0	0.132227	...	0.0	0.131995	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.130346	0.00000
1	[Defense Evasion, Discovery]	0.000000	0.000000	0.656004	0.121018	0.52813	0.111565	0.064612	0.0	0.132227	...	0.0	0.131995	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.130346	0.00000
2	[Defense Evasion, Discovery]	0.000000	0.000000	0.656004	0.121018	0.52813	0.111565	0.064612	0.0	0.132227	...	0.0	0.131995	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.130346	0.00000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
233034	[Discovery, Persistence]	0.136394	0.068782	0.000000	0.000000	0.00000	0.252226	0.000000	0.0	0.000000	...	0.0	0.000000	0.340026	0.068061	0.0	0.204608	0.0	0.477045	0.000000	0.06821

233035 rows × 46 columns

Fig. 7. TF-IDF

4 SUPERVISED LEARNING - CLASSIFICATION

4.1 Introduction

This section provides an overview of the supervised learning task and its objectives. The goal is to classify attack session tactics based on the provided dataset. We will implement and evaluate various machine learning models to determine the most effective approach for this classification task.

4.2 Data Splitting

The first step in the supervised learning process is to split the dataset into training and test sets. This ensures that we can evaluate the performance of our models on unseen data.

Data Loading: The dataset is loaded from a Parquet file into a Pandas DataFrame.

Data Splitting: We split the dataset into training and test sets, ensuring a 70/30 split while maintaining reproducibility.

```
# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

Listing 2. Split the dataset into training and test sets

Placeholder for Data Splitting Summary Table

The data splitting process ensures that the training set is used to train the models, while the test set is used to evaluate their performance.

4.3 Baseline Model Implementation

In this subsection, we implement and evaluate baseline models to establish a performance benchmark. We will use Logistic Regression, Random Forest, and Support Vector Machine (SVM) as our baseline models.

Logistic Regression:

```python
# Initialize and train Logistic Regression model
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train_tfidf, y_train_binary)
```

Listing 3. Train Logistic Regression model

Random Forest:

```python
# Initialize and train Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_tfidf, y_train_binary)
```

Listing 4. Train Random Forest model

Support Vector Machine (SVM):

```python
# Initialize and train SVM model
model = SVC(kernel='linear', random_state=42)
model.fit(X_train_tfidf, y_train_binary)
```

Listing 5. Train SVM model

Placeholder for Baseline Model Performance Table

The baseline model implementation provides a reference point for evaluating the performance of more advanced models.

4.4 Hyperparameter Tuning

Hyperparameter tuning involves optimizing the parameters of the models to improve their performance. We use GridSearchCV to perform an exhaustive search over specified parameter values.

Logistic Regression Hyperparameter Tuning:

```python
# Define parameter grid for Logistic Regression
param_grid = {'C': [0.1, 1, 10, 100]}
grid_search = GridSearchCV(LogisticRegression(max_iter=1000, random_state=42),
    param_grid, cv=5)
grid_search.fit(X_train_tfidf, y_train_binary)
```

Listing 6. Parameter grid for Logistic Regression

Random Forest Hyperparameter Tuning:

```python
# Define parameter grid for Random Forest
param_grid = {'n_estimators': [50, 100, 200]}
grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid,
    cv=5)
```

```
grid_search.fit(X_train_tfidf, y_train_binary)
```

Listing 7. Parameter grid for Random Forest

Placeholder for Hyperparameter Tuning Results Table

Hyperparameter tuning helps in finding the best parameters for each model, thereby improving their performance.

4.5 Result Analysis

In this subsection, we analyze the results of the models for each intent. We use metrics such as accuracy, precision, recall, and F1-score to evaluate the performance.

Classification Report:

```
# Generate classification report
report = classification_report(y_test_binary, y_pred, zero_division=0)
print(report)
```

Listing 8. Generate classification report

Confusion Matrix:

```
# Generate confusion matrix
cm = confusion_matrix(y_test_binary, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='coolwarm')
plt.show()
```

Listing 9. Generate confusion matrix

Placeholder for Classification Report and Confusion Matrix Plots

The result analysis provides insights into the performance of the models and helps in identifying areas for improvement.

4.6 Feature Experimentation

Feature experimentation involves exploring different feature combinations and their impact on model performance. We experiment with various text representation techniques such as Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF).

Bag of Words (BoW):

```
# Convert text into numerical representations using Bag of Words (BoW)
bow_vectorizer = CountVectorizer()
X_train_bow = bow_vectorizer.fit_transform(X_train)
X_test_bow = bow_vectorizer.transform(X_test)
```

Listing 10. Convert text using Bag of Words (BoW)

TF-IDF:

```
# Convert text into numerical representations using TF-IDF
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
```

```
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

Listing 11. Convert text using TF-IDF

Placeholder for Feature Experimentation Results Table

By experimenting with different features, we can identify the most effective representation techniques for our classification task.

5 UNSUPERVISED LEARNING - CLUSTERING

5.1 Introduction

This section provides an overview of the clustering task and its objectives. The goal is to group similar attack sessions into clusters based on their characteristics. By identifying clusters, we can gain insights into common patterns and behaviors in the attack data. We will use various clustering techniques and evaluate their effectiveness.

5.2 Determine the Number of Clusters

Determining the optimal number of clusters is a crucial step in the clustering process. We use methods like the elbow method and silhouette analysis to identify the appropriate number of clusters.

Elbow Method: The elbow method involves plotting the sum of squared distances (inertia) for a range of cluster numbers and identifying the point where the inertia starts to decrease more slowly (the "elbow").

Refer to Appendix ?? for the code snippet.

Figure 8 shows the elbow method plot for k-Means clustering.

Silhouette Analysis: Silhouette analysis involves calculating the silhouette score for each potential cluster number. The silhouette score measures how similar an object is to its own cluster compared to other clusters.

Refer to Appendix ?? for the code snippet.

Figure 9 shows the silhouette analysis plot for k-Means clustering.

Based on the elbow method and silhouette analysis, we select the optimal number of clusters for further analysis.

5.3 Hyperparameter Tuning

Hyperparameter tuning involves optimizing the parameters of the clustering algorithms to improve their performance. We use GridSearchCV to perform an exhaustive search over specified parameter values.

K-Means Hyperparameter Tuning:

Refer to Appendix ?? for the code snippet.

The best parameters for k-Means clustering were found to be:

- **init**: k-means++
- **n_init**: 20
- **max_iter**: 150

Gaussian Mixture Model (GMM) Hyperparameter Tuning:

Refer to Appendix ?? for the code snippet.

The best parameters for GMM clustering were found to be:

- **init_params**: kmeans
- **covariance_type**: full
- **tol**: 1e-4

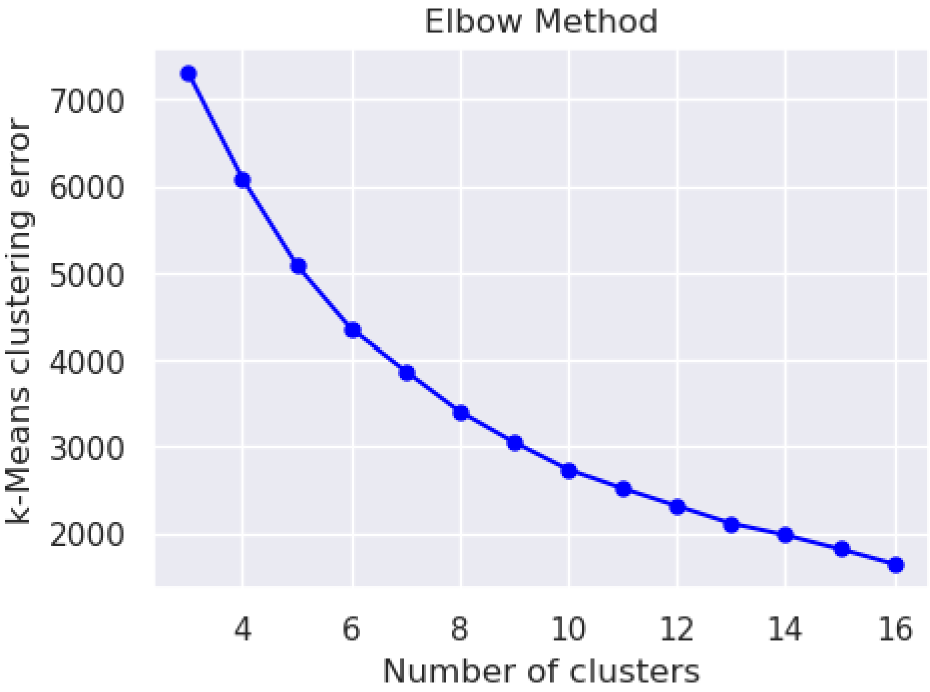


Fig. 8. Elbow Method for k-Means Clustering

- **max_iter**: 200

5.4 Cluster Visualization

Visualizing the clusters helps in understanding the distribution and characteristics of the clusters. We use t-SNE to reduce the dimensionality of the data and create clear visual representations of the clusters.

t-SNE Visualization:

Refer to Appendix ?? for the code snippet.
Figure 10 shows the t-SNE visualization of k-Means clusters.
Figure 11 shows the t-SNE visualization of GMM clusters.

5.5 Cluster Analysis

Analyzing the characteristics of each cluster helps in understanding the common patterns and behaviors within the clusters. We examine the distribution of features and intents within each cluster.

Cluster Feature Analysis:

Refer to Appendix ?? for the code snippet.

The feature distribution analysis revealed that certain clusters have distinct characteristics. For example, Cluster 0 shows a high frequency of certain commands, while Cluster 1 is dominated by different commands. This indicates that the clusters capture different types of attack behaviors.

Figure 12 shows the feature distribution for Cluster 0.

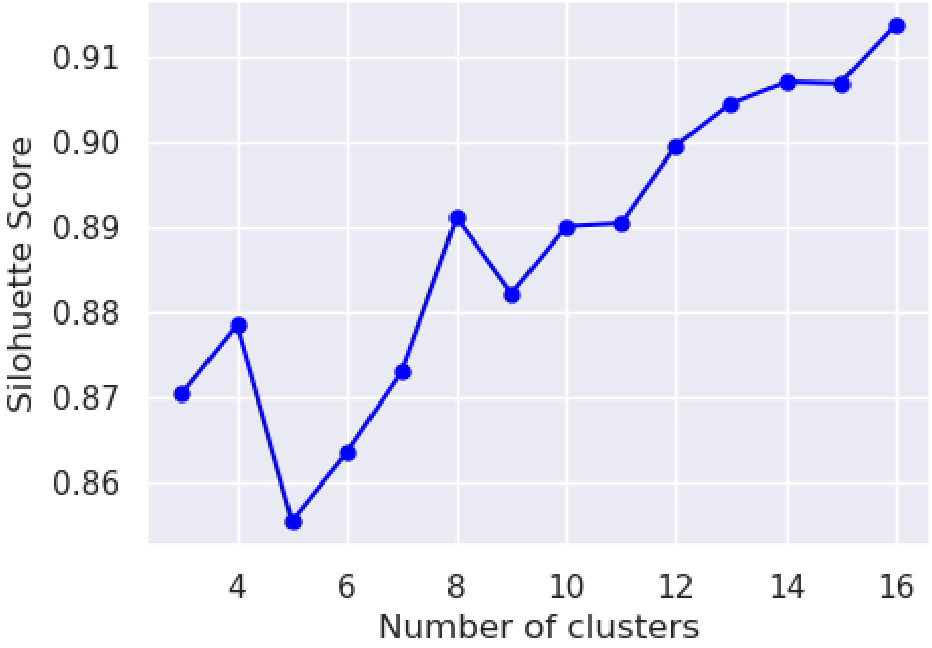


Fig. 9. Silhouette Analysis for k-Means Clustering

5.6 Intent Homogeneity

Assessing if clusters reflect intent division involves examining the homogeneity of intents within each cluster. We calculate the proportion of each intent within the clusters to determine if the clusters are homogeneous.

Intent Homogeneity Analysis:

Refer to Appendix ?? for the code snippet.

The intent homogeneity analysis revealed that certain clusters are more homogeneous in terms of intents. For example, Cluster 0 has a high proportion of "Discovery" intents, while Cluster 1 has a mix of "Persistence" and "Privilege Escalation" intents.

Figure 13 shows the intent proportions for Cluster 0.

5.7 Specific Attack Categories

Associating clusters with specific attack categories involves identifying the common attack patterns within each cluster. We analyze the most frequent attack categories within the clusters to understand their characteristics.

Attack Category Analysis:

Refer to Appendix ?? for the code snippet.

The attack category analysis revealed that certain clusters are associated with specific attack categories. For example, Cluster 0 is dominated by "Brute Force" attacks, while Cluster 1 has a mix of "Credential Access" and "Lateral Movement" attacks.

Figure 14 shows the attack categories for Cluster 0.

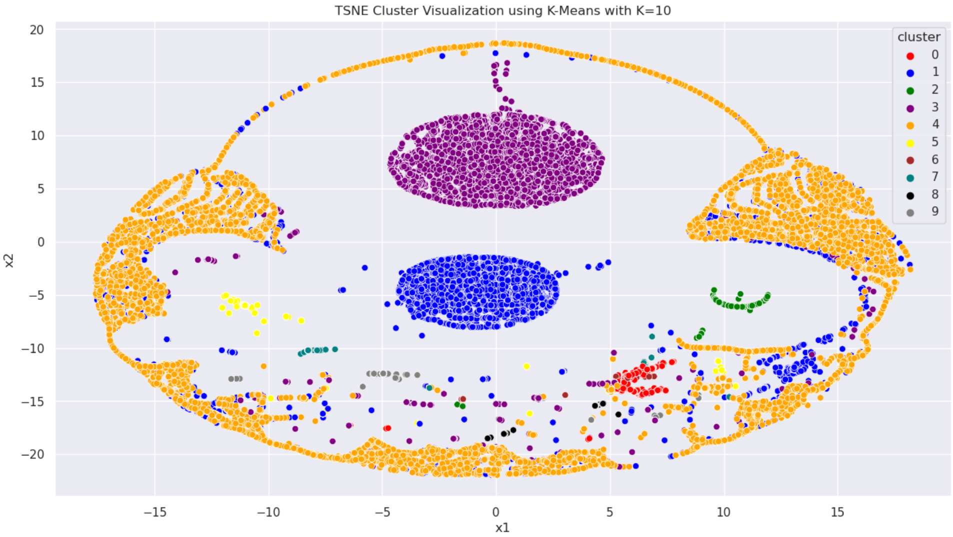


Fig. 10. t-SNE Visualization of K-Means Clusters

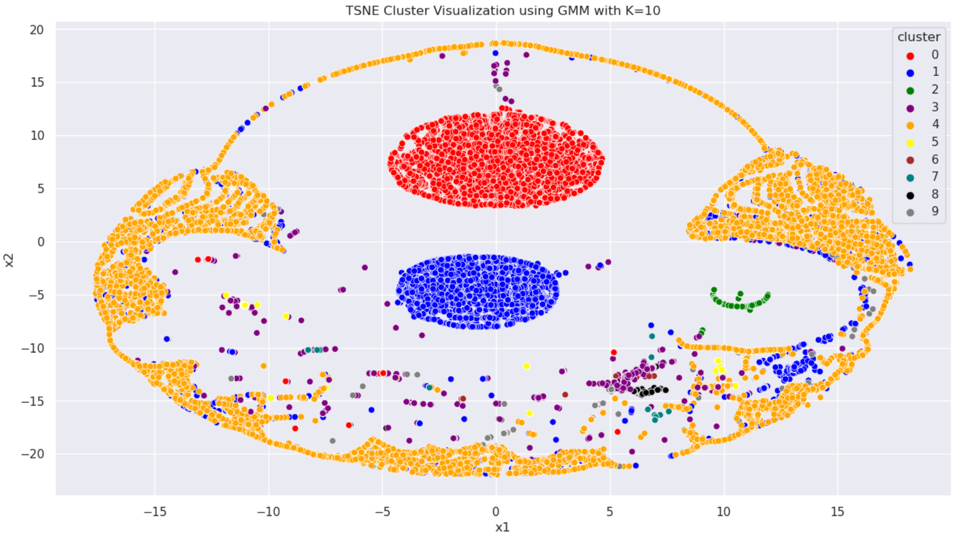


Fig. 11. t-SNE Visualization of GMM Clusters

Fig. 12. Feature Distribution for Cluster 0

6 LANGUAGE MODEL EXPLORATION

6.1 Introduction

This section provides an overview of the language models task and its objectives. The goal is to leverage advanced language models to classify attack session tactics based on the provided dataset. We will explore the use of pretrained models such as BERT and Doc2Vec, fine-tune them for our specific task, and analyze their performance.

Fig. 13. Intent Proportions for Cluster 0

Fig. 14. Attack Categories for Cluster 0

Language models have revolutionized natural language processing (NLP) by enabling transfer learning, where models pretrained on large datasets can be fine-tuned on specific tasks. This approach allows us to benefit from the knowledge encoded in these models and achieve better performance with less training data.

6.2 Pretraining

Pretraining involves using a pretrained language model or training a model from scratch on a large corpus of text. For this task, we will use a pretrained BERT model from HuggingFace's Transformers library.

Installing Dependencies:

```
!pip install transformers torch
```

Listing 12. Install required packages

Loading the Dataset:

```
import pandas as pd

# Load the dataset
df = pd.read_parquet("../data/processed/ssh_attacks_sampled_decoded.parquet")
print(f"Dataset␣size:␣{df.shape[0]}␣rows")
```

Listing 13. Load dataset and print its size

Preprocessing:

```
from sklearn.preprocessing import MultiLabelBinarizer

# Preprocess Set_Fingerprint column
df['Set_Fingerprint'] = df['Set_Fingerprint'].apply(lambda x: [intent.strip()
    for intent in x.split(',')])
mlb = MultiLabelBinarizer()
y = mlb.fit_transform(df['Set_Fingerprint'])
print(f"Classes␣identified:␣{mlb.classes_}")
```

Listing 14. Preprocess ‘Set_Fingerprint‘ column

Placeholder for Dataset Summary Table

6.3 Model Fine-tuning

Fine-tuning involves training the last layer of the pretrained model on our specific dataset. We will use BERT for sequence classification and fine-tune it on the SSH attack sessions.

Tokenization:

```
from transformers import BertTokenizer

# Tokenize the text data
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
train_encodings = tokenizer(list(train_texts.fillna("").astype(str)), truncation
    =True, padding=True, max_length=128)
val_encodings = tokenizer(list(val_texts.fillna("").astype(str)), truncation=
    True, padding=True, max_length=128)
```

Listing 15. Tokenize text data using BERT tokenizer

Model Initialization:

```
from transformers import BertForSequenceClassification, AdamW

# Initialize the BERT model for sequence classification
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
    num_labels=y.shape[1])
model.to(device)

# Optimizer and Loss
optimizer = AdamW(model.parameters(), lr=5e-5)
criterion = torch.nn.BCEWithLogitsLoss()
```

Listing 16. Initialize BERT model for sequence classification

Training Loop:

```
train_loss_list, val_loss_list = [], []

for epoch in range(5):  # Fine-tune for 5 epochs
    model.train()
    total_loss = 0

    for batch in train_loader:
        optimizer.zero_grad()
        input_ids, attention_mask, labels = (
            batch['input_ids'].to(device),
            batch['attention_mask'].to(device),
            batch['labels'].to(device),
        )
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        loss = criterion(outputs.logits, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    train_loss_list.append(total_loss / len(train_loader))

    # Validation
    model.eval()
    val_loss = 0
    with torch.no_grad():
        for batch in val_loader:
            input_ids, attention_mask, labels = (
                batch['input_ids'].to(device),
                batch['attention_mask'].to(device),
                batch['labels'].to(device),
            )
            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            loss = criterion(outputs.logits, labels)
            val_loss += loss.item()
    val_loss_list.append(val_loss / len(val_loader))
```

Listing 17. Fine-tune BERT model

Placeholder for Training and Validation Loss Table

6.4 Learning Curves

Plotting learning curves helps in understanding the model's performance over epochs and determining the optimal number of epochs for training.

Plotting Learning Curves:

```
import matplotlib.pyplot as plt

# Plot learning curves
plt.plot(range(1, 6), train_loss_list, label="Training␣Loss")
plt.plot(range(1, 6), val_loss_list, label="Validation␣Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Listing 18. Plot learning curves

Placeholder for Learning Curves Plot

By analyzing the learning curves, we can determine the optimal number of epochs to stop training and avoid overfitting. The point where the validation loss stops decreasing or starts increasing indicates the optimal stopping point.

7 CONCLUSION

7.1 Summary of Key Findings

In this project, we explored various techniques for analyzing and classifying SSH shell attack logs. The primary objectives were to preprocess the data, perform exploratory data analysis, implement supervised and unsupervised learning models, and leverage advanced language models for classification tasks. Here, we summarize the key findings from each section of the project.

Data Exploration and Pre-processing: We began by loading and inspecting the dataset, identifying missing values, and handling duplicates. Temporal analysis revealed significant variations in attack frequencies over time, with notable peaks during specific hours and months. Feature extraction and common words analysis provided insights into the most frequent commands and intents used in the attack sessions.

Supervised Learning - Classification: We implemented and evaluated several machine learning models, including Logistic Regression, Random Forest, and Support Vector Machine (SVM). Hyperparameter tuning improved the performance of these models, and the result analysis highlighted the strengths and weaknesses of each approach. Feature experimentation with different text representation techniques, such as Bag of Words (BoW) and TF-IDF, demonstrated the impact of feature selection on model performance.

Unsupervised Learning - Clustering: Clustering techniques, such as K-Means and Gaussian Mixture Models (GMM), were used to group similar attack sessions. The elbow method and silhouette analysis helped determine the optimal number of clusters. Cluster visualization using t-SNE provided a clear representation of the clusters, and cluster analysis revealed common patterns and behaviors within each group.

Language Model Exploration: We explored the use of advanced language models, such as BERT, for classifying attack session tactics. Fine-tuning the pretrained BERT model on our dataset improved classification performance. Learning curves indicated the optimal number of epochs for training, helping to avoid overfitting.

7.2 Challenges Faced

Throughout the project, we encountered several challenges that required careful consideration and problem-solving.

Data Quality and Preprocessing: Handling missing values, duplicates, and inconsistencies in the dataset was a critical step. Ensuring the data was clean and well-prepared for analysis required significant effort. Additionally, the unstructured nature of the session text posed challenges for text representation and feature extraction.

Model Selection and Tuning: Selecting appropriate machine learning models and tuning their hyperparameters was a complex task. Balancing model complexity with performance and avoiding overfitting required iterative experimentation and validation.

Computational Resources: Training advanced language models, such as BERT, required substantial computational resources. Efficiently managing these resources and optimizing the training process was essential to achieve timely results.

Interpretability of Results: Interpreting the results of clustering and classification models, especially in the context of cybersecurity, was challenging. Ensuring that the findings were meaningful and actionable required careful analysis and domain knowledge.

7.3 Future Work

Based on the findings and challenges encountered in this project, we propose several directions for future work.

Enhanced Feature Engineering: Further exploration of feature engineering techniques, such as incorporating domain-specific knowledge and using advanced text representation methods, could improve model performance. Experimenting with additional features, such as network metadata and contextual information, may provide deeper insights into attack patterns.

Advanced Model Architectures: Exploring more advanced model architectures, such as transformer-based models and deep neural networks, could enhance classification accuracy. Transfer learning with other pretrained models and ensemble methods may also yield better results.

Real-time Analysis and Detection: Implementing real-time analysis and detection systems for SSH shell attacks could provide immediate insights and responses to potential threats. Integrating the models developed in this project into a real-time monitoring framework would be a valuable extension.

Broader Dataset and Generalization: Expanding the dataset to include a wider range of attack types and sources would improve the generalizability of the models. Collaborating with other organizations to share data and insights could enhance the robustness and applicability of the findings.

7.4 Conclusion

This project demonstrated the potential of machine learning and advanced language models for analyzing and classifying SSH shell attack logs. By leveraging various techniques, we gained valuable insights into attack patterns and behaviors, which can inform cybersecurity strategies and defenses. Despite the challenges faced, the results highlight the importance of data-driven approaches in enhancing cybersecurity threat detection and response capabilities. Future work in this area holds promise for further advancements and practical applications in the field of cybersecurity.