Final report- Web application attacks

Shalev Mazuz 318885712

Part 1: General Information

1. I would like to detect **Web Application Attacks** like brute force attacks, SQL injections and cross-site scripting (XSS).
2. The typical characteristics of attacks in this category are targeting user input points, injection of malicious code and bypassing authentication or access controls in order to gain unauthorized access and obtain confidential information.
3. Brute force attacks in the records will appear as numerous short flows with high packet rates, small packet sizes, and a consistent source IP, reflecting repeated login attempts to gain unauthorized access.
   XSS and SQL injection attacks will be characterized by larger forward packet sizes due to the transmission of substantial code payloads to the backend. These attacks often show longer flow durations, as the server requires additional time to execute SQL queries or return HTML content in response.

Part 2: The data-set and feature extraction

1. The data set I used is **CICIDS2017**:
   a. https://www.unb.ca/cic/datasets/ids-2017.html
   b. The CICIDS2017 dataset was generated by the Canadian Institute for Cybersecurity over five consecutive days in a controlled network environment that simulated real-world enterprise traffic. It included a mix of benign user activities (browsing, emailing, file transfers, streaming) and attack scenarios (e.g., DoS, DDoS, brute force, SQL injection, XSS, botnet, port scanning). Traffic was captured in PCAP format and processed using CICFlowMeter to extract over 80 bidirectional flow features, such as packet counts, byte counts, durations, and timing metrics. Each flow was then labelled according to the type of activity or attack being performed during that time window.
   c. The dataset is organized into folders containing raw PCAP files for each day, corresponding CSV files with extracted flow-based features, and a combined machine learning–ready CSV with all labelled records. Each day's files represent different scenarios: Monday (benign only), Tuesday (FTP/SSH brute force), Wednesday (DoS attacks), Thursday (web attacks including brute force, SQL injection, and XSS, plus infiltration), and Friday (DDoS and botnet). The CSV files contain around 80 features along with a label indicates whether the flow is benign or attack. As it can be seen, only the Thursday file is relevant for this attack category.
2. Since there are almost 80 features, selecting only few of them is unlikely to provide enough data to classify flows. Thus, I used automatic selection with *SelectKBest* function from *sklearn.feature_selection* library to select the most

important features. For all 3 methods I selected 50 features. I will represent the top 5 features selected for each method.

<u>Random Forest model:</u>

Random Forest classifier has a built-in features importance ranking. These were the 5 most important features for this model:

1. **Init_win_bytes_backward:** the total number of bytes sent in the initial window in the reverse direction (from the destination to the source). This feature may help distinguish attacks from benign flow since brute force attacks typically have lower values due to small, repetitive requests, while SQL injection tends to show higher and more variable values from abnormal server responses.
2. **Max_packet_length:** represents the length (in bytes) of the largest packet observed in a flow. brute force attacks usually involve short, repeated requests resulting in small maximum packet sizes, while SQL injection or XSS may cause larger response payloads or errors, leading to larger packets. As a result, this feature helps identify anomalous flows by capturing extremes in data size.
3. **Average_packet_size:** measures the mean size of packets in a network flow. Brute force attacks often involve many small, uniform packets, leading to a lower average size, whereas SQL injection or XSS attacks may produce more variable and larger packet sizes.
4. **Total_length_of_fwd_packets:** captures the total size (in bytes) of all packets sent from the source to the destination in a flow. Brute force attacks typically have small, repetitive requests resulting in low total forward traffic, while SQL injection or XSS attacks might send larger or more complex payloads, increasing the total forward packet length.
5. **Packet_length_mean:** represents the average length of all packets in a flow, combining both forward and backward directions. Brute force attacks often have consistently short packets, resulting in a low mean, while SQL injection or XSS attacks may cause more varied or larger packets, increasing the average.

<u>KNN model:</u>

For the KNN model I did SHAP[1] analysis to evaluate features importance. These are the most important features according to the SHAP analysis:

1. **Init_win_bytes_backward:** described before for the Random Forest.
2. **Init_win_bytes_forward:** represents the initial TCP window size (in bytes) advertised by the client during the handshake. Normal browsers typically use standard values (e.g., 64240 or 65535), while many attack tools use different or smaller defaults (e.g., 29200 or 8192). These differences arise because automated tools often rely on non-browser TCP/IP stacks or virtualized environments.
3. **Min_seg_size_forward:** represents the smallest TCP payload size (in bytes) sent by the client during a flow. SHAP analysis shows that higher values often push the classifier toward attack labels. This happens because attacks such as SQL injection and XSS generate large request segments, whereas normal browsing tends to include at least one small packet (e.g., login attempt).

4. **Flow IAT mean:** represents the average inter-arrival time between packets in a flow. SHAP analysis shows that low values push the model toward benign classification, while higher values have less influence. This reflects the fact that normal web traffic tends to produce short, frequent packet exchanges (low IAT), whereas attack flows are more varied, some having higher inter-arrival times due to irregular or payload-heavy traffic. As a result, the model interprets "low mean IAT" as a benign signal, but does not treat "high mean IAT" as a definitive attack signal.
6. **Destination port:** represents the TCP or UDP port number on the server side (destination) of the connection. SHAP analysis shows that higher port values tend to push the model toward benign classification, while low, well-known ports are associated with attack traffic. This pattern arises because most attacks in the dataset target specific low ports (80), whereas benign traffic often uses higher, ephemeral ports.

   <u>SVM model:</u>
   For the SVM model I did SHAP[2] analysis to evaluate features importance. These are the most important features according to the SHAP analysis:

1. **Min_seg_size_forward:** described before for KNN.
2. **Init_win_bytes_backward:** described before for Random Forest.
3. **Destination port:** described before for KNN.
4. **Min_packet_length:** represents the smallest packet size observed in a flow. SHAP analysis shows that lower values push the model toward attack classification, reflecting the fact that malicious traffic often contains extremely small packets, such as SQL injection or XSS probes or brute-force attempts. However, single normal login packets may also be small, so min_packet_length alone is not definitive. The model uses it in combination with other features to distinguish repeated or patterned small-packet traffic typical of attacks from occasional small packets in benign user activity.
5. **Bwd_packet_length_min:** represents the smallest packet size sent by the server in a flow. SHAP analysis shows that lower values push the model toward attack classification, reflecting that attacks often elicit very small server responses, such as login failure messages or error replies to probes. While normal traffic can also contain small packets (e.g., acknowledgments), extremely low values combined with other flow characteristics might signal malicious activity.

<u>Part 3: Implemented ML Models</u>

All the models I used were implemented with the scikit-learn (sklearn) Python library, which offers a wide range of machine learning tools.
All model implementations followed a similar approach:
First, I performed some preprocessing on the data. The original dataset had a multiclass label, but due to the very small number of samples in some classes, the model couldn't recognize them properly. To address this issue, I converted the problem to binary classification, each row is now labelled as either BENIGN or ATTACK.

Next, I removed rows containing unconventional values such as infinity or NaN. After that, I scaled the features, since features with larger numerical ranges can dominate the learning process if left unscaled.
Then, I performed feature selection as described before. At this point, I had a clean set of features and their corresponding labels.
I split the data into training and test sets using an 80-20 ratio, 80% of the data was used to train the model, and the remaining 20% was used to test it.
Finally, I trained the model using the fit(X_train, y_train) function, specifying the exact parameters, which will be described in the following section, and the predict(X_test) function to evaluate it on the test set.

KNN

1. KNN requires setting 2 main parameters:
    a. Neighbours number (K): This parameter defines how many neighbours are considered when labelling a sample. I conducted the experiment[3] using odd values of K from 1 to 15 (using odd values is important to avoid ties). The best performance was achieved with **K = 3**, while larger values led to a sharp decrease in all metrics, likely because the dataset contains many BENIGN samples, and considering too many neighbours can cause attack samples to be misclassified as benign.
    b. Distance metric: I used the **Euclidean distance**, which is the most commonly applied metric. I also experimented with other distance measures, but the results were nearly identical across all metrics.
2. The training and testing details were the same for all models and have been described earlier.
3. Results:

Label	Precision	Recall	F1-score	Support
ATTACK	0.99	0.99	0.99	436
BENIGN	1.00	1.00	1.00	33611

   Accuracy: 0.9997 (99.97%)
   The confusion matrix[4] shows that the model misclassified 6 BENIGN flows out of 33611 flows, and out of 436 ATTACK flows, it correctly identified 432 as attacks.
4. In the notebook, I printed the scaled feature values for one false positive sample and one false negative sample. Combined with the earlier SHAP analysis, I used this information to investigate why the model misclassified these samples.
   The false positive case, a legitimate benign flow, was incorrectly classified as an attack due to its anomalous feature values. The SHAP analysis indicates that a high value for Init_Win_bytes_backward (2.76) and Init_Win_bytes_forward (1.71) are strong positive contributors to an ATTACK classification. These high values, likely characteristic of a specific legitimate application's communication protocol, caused the KNN model to find its nearest neighbours among attack

flows that share similar initial TCP window sizes. Consequently, the model's distance-based classification mechanism failed to correctly identify the benign nature of the flow, leading to misclassification.
Conversely, the false negative instance, a true attack, was misidentified as benign. This misclassification is attributed to the attack's ability to mimic the features of normal network traffic. The SHAP plot demonstrates that certain features, such as low values for Packet Length Mean, correlate with a BENIGN classification. For this specific attack, high feature values such as Bwd Packet Length Std (2.91), Max Packet Length (1.59), and a normal Down/Up Ratio (0.63) created a feature vector that closely resembled a legitimate data transfer. These values, which are typical of benign traffic, caused the attack's feature vector to be located within a dense cluster of benign data points. The resulting distance calculation led the KNN model to find its nearest neighbours among benign flows, overriding any potential indicators of malicious activity.

SVM

1. Unlike the other two models, SVM's parameters are less intuitive. To find the best combination, I used scikit-learn's GridSearchCV, which trains and evaluates the model with multiple parameter combinations and returns the ones that perform best. I applied 5-fold cross-validation, meaning the data was split and trained five times to ensure a more reliable evaluation. Based on this process, the best parameters, later used for my final model, were:
    a. Kernel: **RBF.** Although this is the default option, it is also the most suitable for this model. The kernel parameter defines how the SVM separates the data, and the RBF kernel is widely used because it can capture complex, non-linear patterns. I also tested the linear kernel, but it produced relatively poor results.
    b. C: **100**. the C parameter controls the balance between having a smooth decision boundary and correctly classifying all training points. A small C allows some misclassifications but creates a simpler, more general boundary. With a large value like C = 100, the model places much more importance on classifying every training point correctly, leading to a stricter and more complex boundary. In the cross-validation search, I tested C values of 10^{-1}, 10^{0}, 10^{1}, 10^{2} and the best performance was achieved with C = 100.
    c. Gamma: **1**. the gamma parameter controls how far the influence of a single training point reaches. A small gamma means each point has a wide influence, leading to a smoother and more general boundary. A larger gamma, such as gamma = 1, makes the model focus more closely on individual points, creating a more detailed and flexible boundary. In the cross-validation search, I tested gamma values of 10^{-3}, 10^{-2}, 10^{-1}, 10^{0} and the best performance was achieved with gamma = 1. I also manually tested gamma values of 'scaled' (the default, gamma = 1 / (n_features *

X.var())) and ‘auto’ (gamma = 1 / n_features), but found that setting gamma = 1 gave better results.

2. After determining the parameters, the training and testing details were the same for all models and have been described earlier.
3. Results:

Label	Precision	Recall	F1-score	Support
ATTACK	0.92	0.98	0.95	436
BENIGN	1.00	1.00	1.00	33611

   Accuracy: 0.9986 (99.86%).
   The confusion matrix[5] shows that the model misclassified 9 attack flows and 37 benign flows.
4. In the notebook, I printed the scaled feature values for one false positive sample and one false negative sample. Combined with the earlier SHAP analysis, I used this information to investigate why the model misclassified these samples.
   The key features of the false positive are the very low scaled value of -1.3046 for Down/Up Ratio and the low scaled value of -0.8356 for Min Packet Length. According to the SHAP analysis, these low feature values have a high negative impact, pushing the model's output towards an ATTACK classification. This pattern is characteristic of certain web attacks or brute-force attempts that involve many small, probing packets with few or no responses from the server. Although this specific sample was benign, its unique configuration such as a single host sending many small requests to a web server without receiving much data back made it appear statistically similar to a malicious scan, leading to the misclassification.
   Conversely, an attack flow was misclassified as benign (a false negative) because it lacked the key identifying features that the model has learned to look for. Although it has a strong negative signal from the ACK Flag Count (1.635), the attack may have been structured in a way that its other features, such as the high min_seg_size_forward (1.009333), had a combined positive influence that outweighed the negative signal from the high ACK Flag Count. For example, an attacker might have executed a slow-and-low brute force attack with a valid looking TCP handshake, which could lead to a mix of both "benign-like" and "attack-like" feature values that led the model to misclassify it.

Part 4: Comparing the results of the 3 AI models

1. Results:

	RF	KNN	SVM
Accuracy	99.98%	99.97%	99.86%
Precision (attack)	1.00	0.99	0.92
Recall (attack)	0.99	0.99	0.98
F1-score (attack)	1.00	0.99	0.95

All models achieved 1.00 score in all metrics for the BENIGN class.

All three models achieved very high accuracy (it should be noted that simply predicting BENIGN would yield 98.7% accuracy), yet their performance on the attack class shows some differences. Random Forest achieved the best results, with perfect precision and F1-score. This is because RF combines many decision trees, which makes it more stable, less sensitive to noise, and better at capturing complex patterns in the data. It also deals well with the imbalance between benign and attack samples, which is important in this dataset.

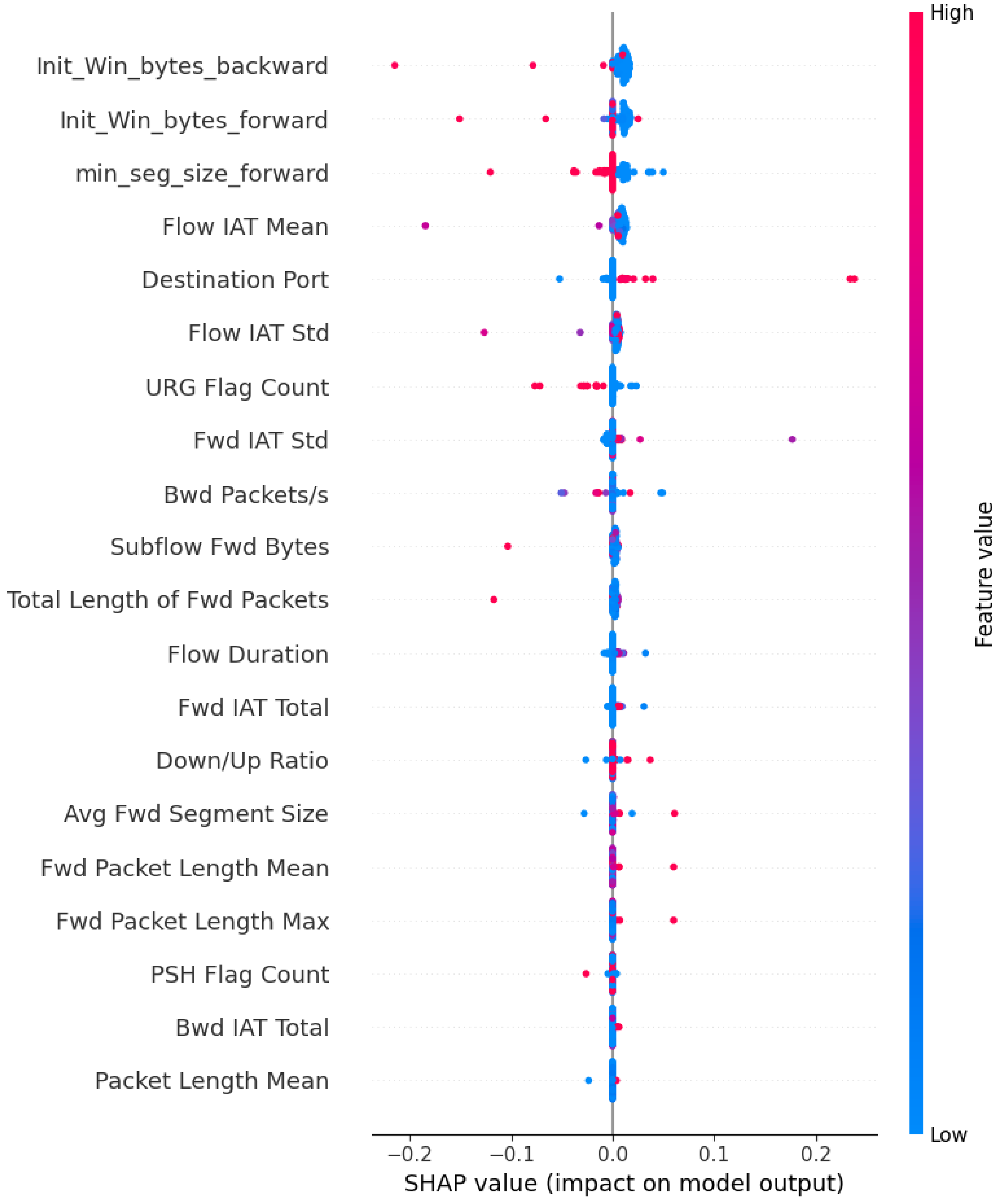
KNN also performed very well, only slightly behind RF. Its strength comes from classifying based on the closest neighbours, which works well when the data has clear clusters. However, in higher dimensions, KNN becomes less reliable, which likely explains the small drop in precision and F1 compared to RF.

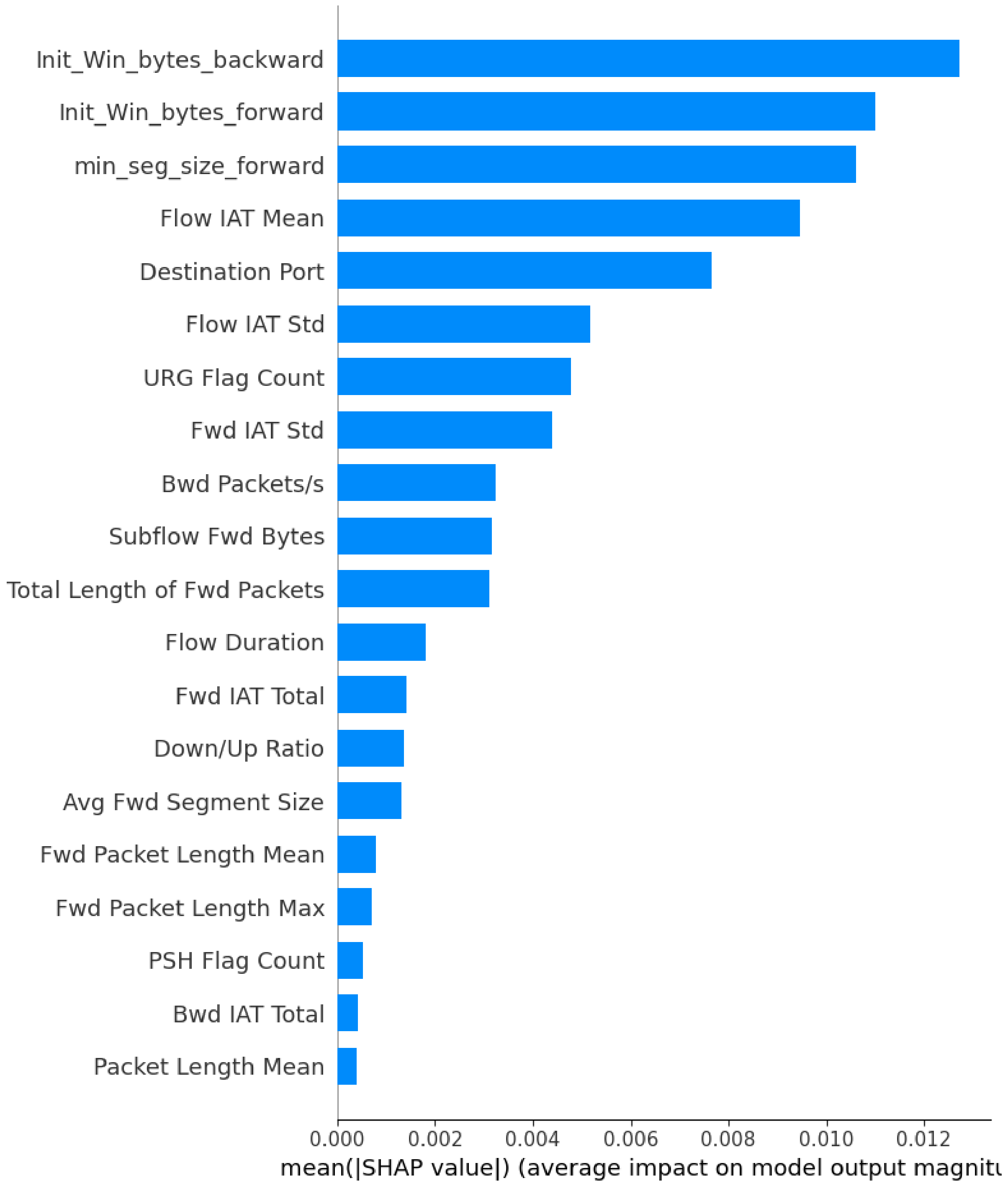
SVM, while still producing strong results, was weaker than the other two. It had lower precision for attacks, meaning it generated more false alarms by labelling benign traffic as malicious. SVM works best when the data is cleanly separated, but in this dataset the overlap between benign and attack samples, combined with the class imbalance, makes it harder for SVM to find an ideal boundary.

2. The summary table[6] shows all the FP and FN group by the models that misclassified them.
   Log events 20000 and 28616 are classified as false negatives by all three models. The models were likely deceived by features with extremely high values, such as Bwd Packet Length Max (4.85 for 20000) and Packet Length Mean (3.05 for 20000, 2.07 for 28616), which are strong indicators of benign activities like file downloads. These BENIGN signals likely overpowered the attack-like features. The models, trained to look for specific attack signatures like high packet rates or small packet sizes, failed to detect these attacks, as they likely represent unusual method of data exfiltration that deviates from typical attack patterns.
   There are few log events that classified correctly by RF but as FP or FN for KNN and SVM. I will focus on log 10359 which classified correctly by RF but is a FP for the two other models.
   Both SVM and KNN were likely misled by the same set of features: a very low Down/Up Ratio (-1.304626) and a low Min Packet Length (-0.835675). These values, as the SHAP analysis suggests, are indicators of an attack, causing the SVM's non-linear decision boundary and the KNN's nearest neighbour search to place the data point into the ATTACK category. However, the Random Forest model correctly classified the flow due to its robust, ensemble-based approach. While some of its decision trees may have been fooled by the low Down/Up Ratio, a majority likely focused on other, more telling features with very high scaled values, such as Init_Win_bytes_backward (2.760234) and PSH Flag Count (1.790541), which are highly indicative of benign traffic. By aggregating the votes from all its individual trees, the Random Forest model was able to reach a correct decision, effectively overcoming the misleading features that caused the misclassification in the other two models.

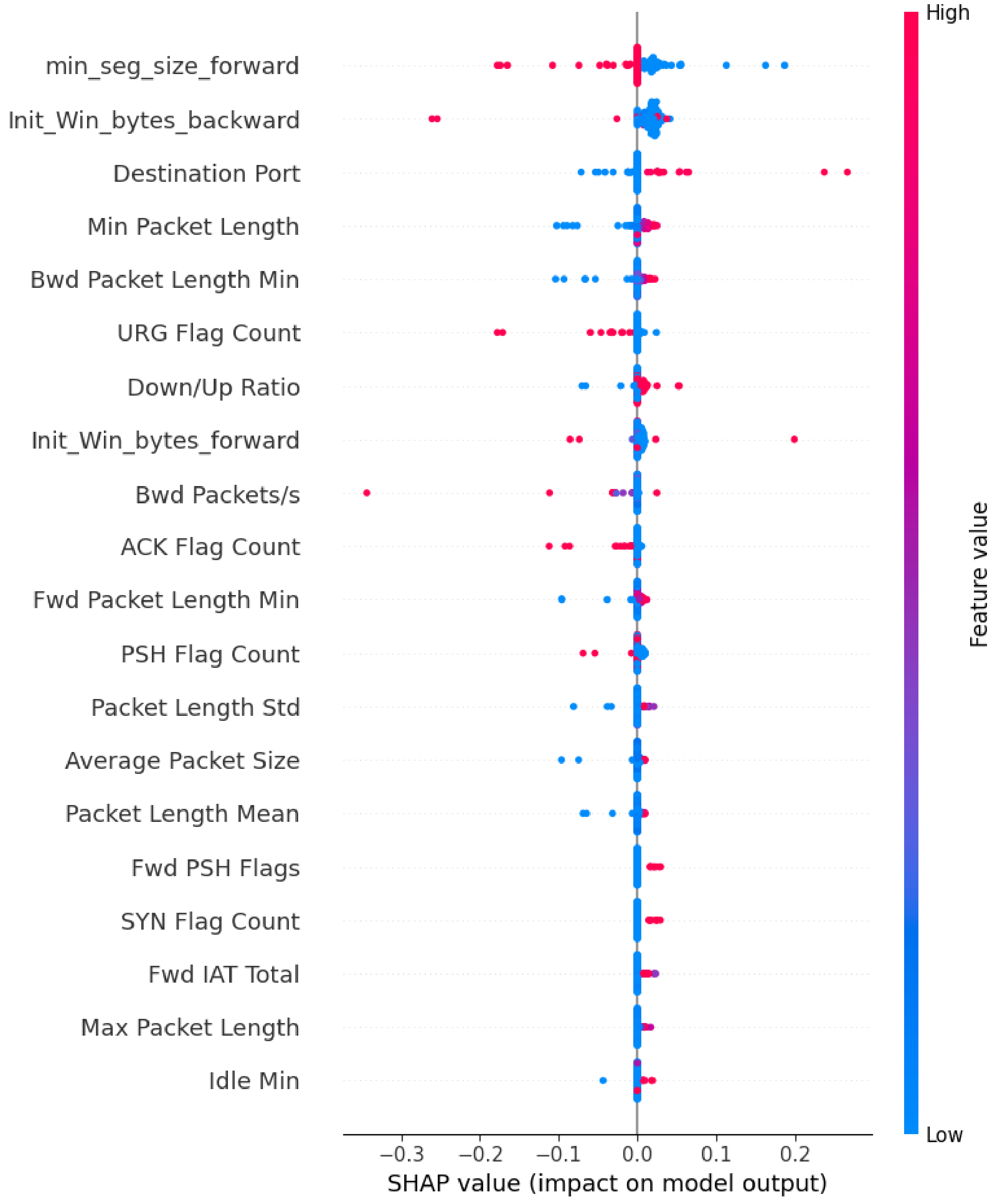
There are 2 log events (2627 and 17955) that classified wrong by RF and SVM but correctly by KNN. This implies that while these logs features might not fit the general pattern of an attack (as learned by SVM and RF), its unique combination of values is still most similar to a small, distinct cluster of other malicious flows in the training data. SVM and Random Forest's general purpose models were over reliant on typical attack patterns and were fooled by an evasive, atypical attack. The KNN, however, succeeded by recognizing a local similarity to a small group of other malicious flows, highlighting the value of its instance-based approach for identifying out of the norm attacks that evade standard detection.

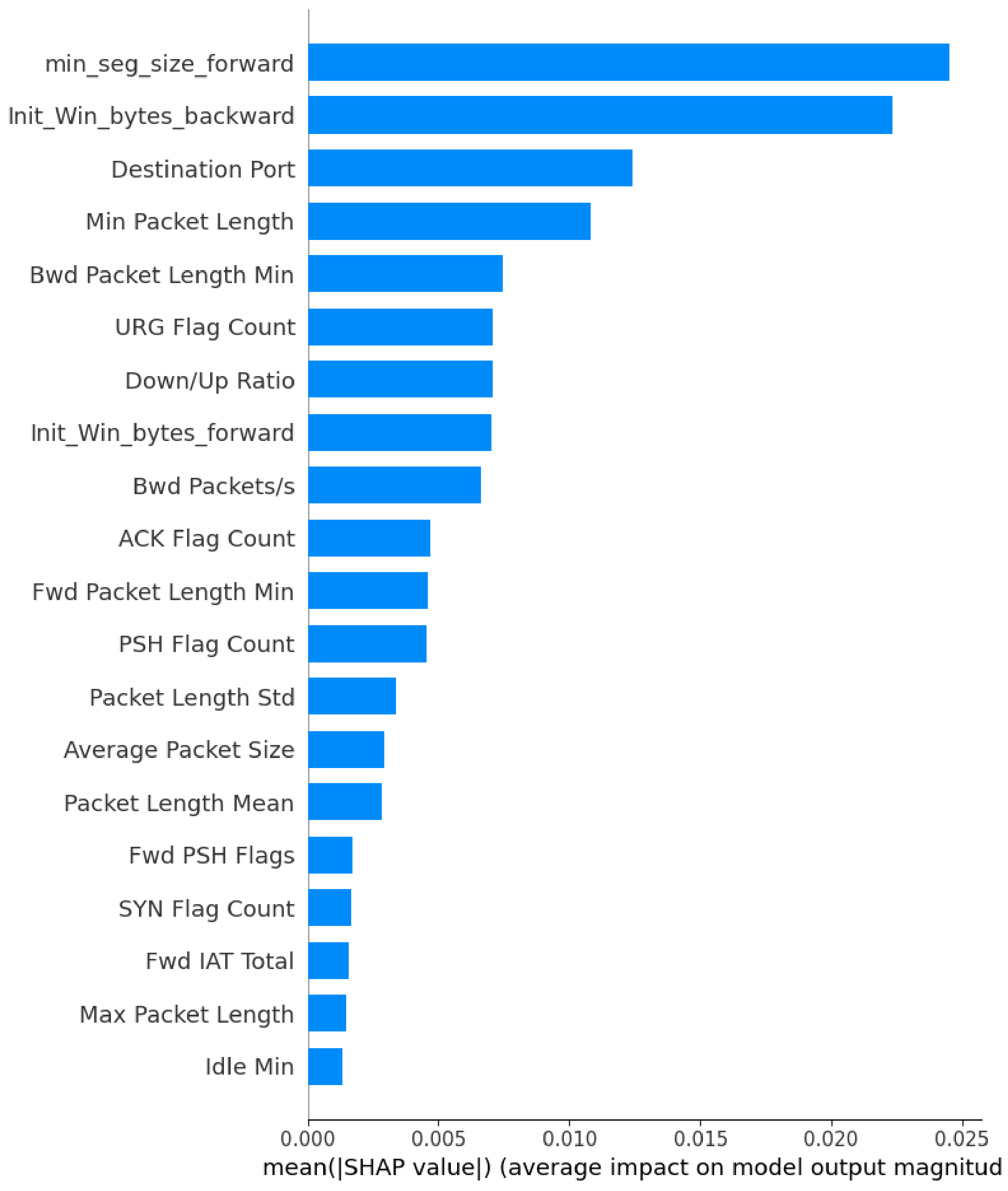
[1] KNN's SHAP analysis:



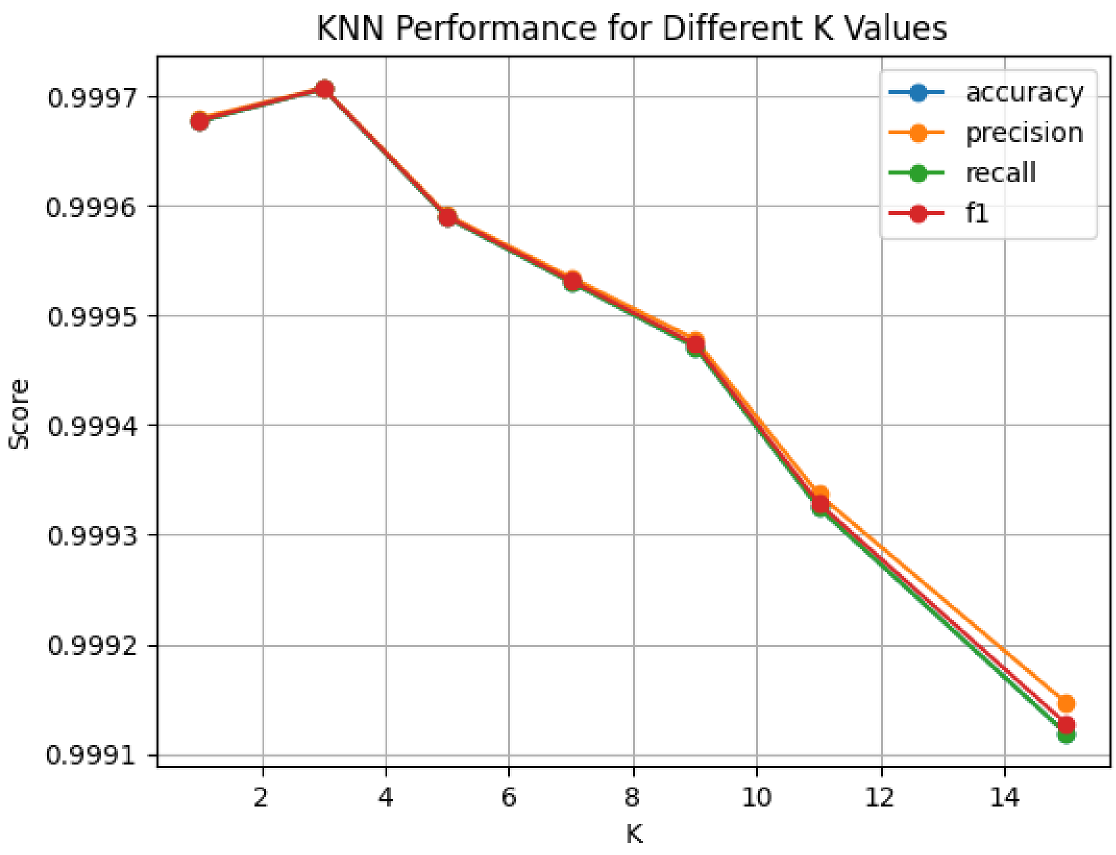

Init_Win_bytes_backward
Init_Win_bytes_forward
min_seg_size_forward
Flow IAT Mean
Destination Port
Flow IAT Std
URG Flag Count
Fwd IAT Std
Bwd Packets/s
Subflow Fwd Bytes
Total Length of Fwd Packets
Flow Duration
Fwd IAT Total
Down/Up Ratio
Avg Fwd Segment Size
Fwd Packet Length Mean
Fwd Packet Length Max
PSH Flag Count
Bwd IAT Total
Packet Length Mean
0.000
0.002
0.004
0.006
0.008
0.010
0.012
mean(|SHAP value|) (average impact on model output magnitu

[2] SVM’s SHAP analysis:

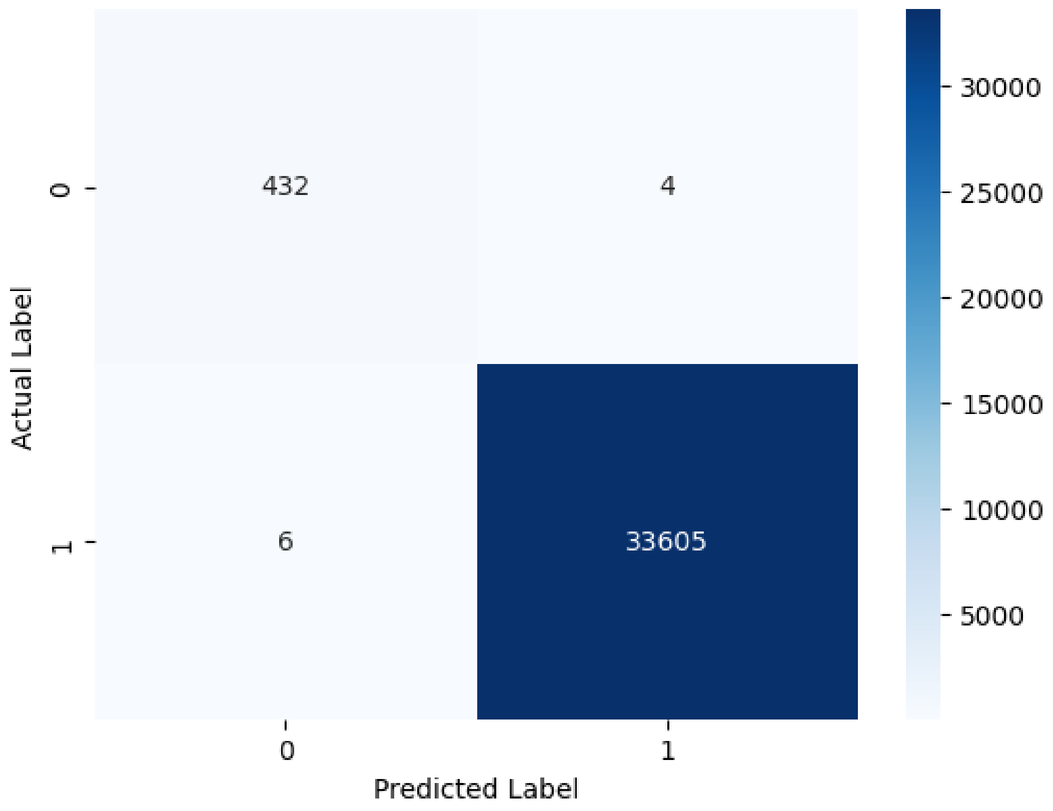



min_seg_size_forward
Init_Win_bytes_backward
Destination Port
Min Packet Length
Bwd Packet Length Min
URG Flag Count
Down/Up Ratio
Init_Win_bytes_forward
Bwd Packets/s
ACK Flag Count
Fwd Packet Length Min
PSH Flag Count
Packet Length Std
Average Packet Size
Packet Length Mean
Fwd PSH Flags
SYN Flag Count
Fwd IAT Total
Max Packet Length
Idle Min
0.000
0.005
0.010
0.015
0.020
0.025
mean(|SHAP value|) (average impact on model output magnitud

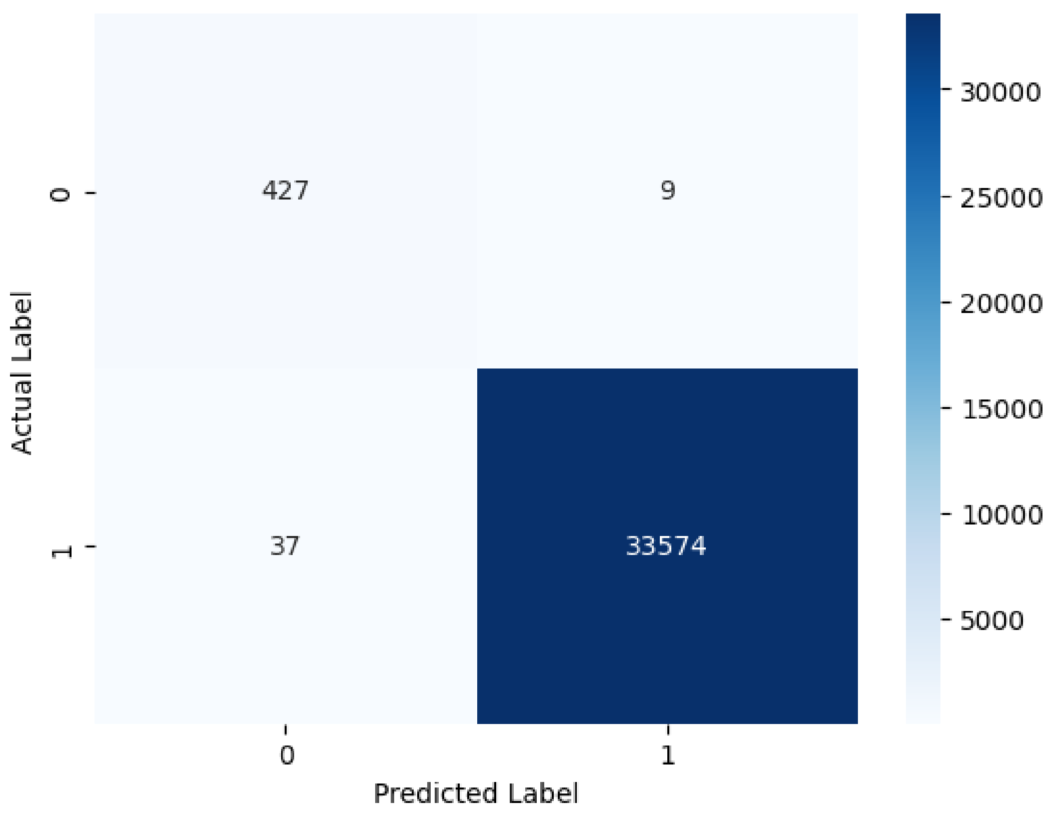
[3] KNN performance for different K values:



[4] KNN confusion matrix:



[5] SVM confusion matrix:



[6] False Positives and False Negatives Summary:

False Positives (FP)

Category	Count	IDs
All 3 models	0	-
Exactly 2 models (KNN+SVM)	5	10359, 19549, 25810, 26523, 31441
Only RF	0	-
Only KNN	1	21062
Only SVM	32	1624, 3058, 3326, 3698, 4692, 6738, 7879, 9200, 9556, 9832, 10684, 11554, 13160, 13369, 14098, 15535, 16102, 16255, 20805, 21060, 21122, 21982, 22220, 22266, 22533, 23486,

		25329, 26113, 28782, 29905, 30687, 32645

False Negatives (FN)

Category	Count	IDs
All 3 models	2	20000, 28616
Exactly 2 models (RF+SVM)	2	2627, 17955
Exactly 2 models (KNN+SVM)	1	6941
Exactly 2 models (RF+KNN)	0	-
Only RF	0	-
Only KNN	1	25029
Only SVM	4	11188, 13460, 14092, 26221