# Intrusion Detection for Wi-Fi networks

## Authors

Joseph Vinson - 101126637

Jake Jazokas - 101083496

AJ Ricketts - 101084146

## Summary of Work

Jake Jazokas - Implemented the majority of the codebase for this project, trained and tested the models, created visual results for the report and wrote the following sections of the final paper: simulation, algorithm description, and results. Also, recorded and edited the final presentation and helped to write the project proposal.

AJ Ricketts - Implemented the live capture of packets using monitor mode, saving and loading the trained models, and wrote the following sections of the final paper: development technologies, abstract, and conclusion. Also helped record the final presentation and helped to write the project proposal.

Joseph Vinson - Wrote the following sections of the final paper: domain and problem, literature review, and methodology. Also helped to record and edit the final presentation and helped to write the project proposal. Helped determine the probability calculations for the extracted features.

# Abstract

With the internet rapidly expanding over the last few decades, and the development and adoption of wireless devices, keeping these devices secure has been a focal point for researchers over recent years. With Wi-Fi security still being a very relevant problem, there have been many different approaches to securing Wi-Fi networks, this includes Wi-Fi intrusion detection, which will be the focal point of this paper. With many different implementations for intrusion detection, it raised the question of which method is the most successful at securing a Wi-Fi network. Although not every method could be compared, this report will outline 3 separate implementations completed by various researchers. After getting a brief overview of all three approaches, one of these will be implemented by us, the results of which will be compared to its respective paper to determine if the approach is feasible in real-world settings. Our implementation will be based on the "*WIDS: An Anomaly Based Intrusion Detection System for Wi-Fi (IEEE 802.11) Protocol*" research paper **[1]**. As in the paper, our implementation will model the normal behaviour of the Wi-Fi protocol, using n-grams, and use machine learning models to classify Wi-Fi traffic flows as normal or malicious **[1]**. After training our model and testing against various attacks, it is clear we maintain a high degree of detection accuracy, with minimal false positives.

# Domain and Problem

Intrusion detection in Wi-Fi networks aims to ensure that all connections within the network come from authorized entities. Our project of Intrusion Detection for WI-Fi networks fits in the overall domain of Wi-Fi security. If we were to give a precise definition for intrusion detection for Wi-fi networks, it would be the prevention of unauthorized damage to computers or data using wireless networks. Wi-Fi security deals with data confidentiality and integrity. To ensure network security, the validity of the connections must be established.

In our project, we will be focusing on the specific Wi-Fi security problem of entity authentication and ensuring that even if an unauthorized entity gains access to the network, they will be recognized promptly and not be allowed to gain further access. Our selected paper attempts to solve the wireless intrusion detection problem by finding the best machine learning model to accurately predict the normalcy of network traffic, using n-gram representations of data.

# Literature Review

The papers used to solve our research questions were, "*WIDS: An Anomaly Based Intrusion Detection System for Wi-Fi (IEEE 802.11) Protocol*" **[1]**, "*Design and Implementation of an Intrusion Prevention System for Wi-Fi Networks 802.11 AC. Mobile, Secure, and Programmable Networking*" **[2]**, and "*A Machine Learning Based Two-Stage Wi-Fi Network Intrusion Detection System*" **[3]**. We used the algorithm described in the first paper mentioned above for the implementation of our project as it provided the highest accuracy when predicting networking intrusions.

The second paper, "*Design and Implementation of an Intrusion Prevention System for Wi-Fi Networks 802.11 AC. Mobile, Secure, and Programmable Networking*" **[2]**, focused on the Bayesian game model to make predictions about network activity. The Bayesian game model is dynamic and each player selects their behaviour depending on the system's current state and the amount of information said player has **[2]**. The cost function of the model is applied until the system reaches equilibrium.

When equilibrium is reached, the probabilities can be derived using partial derivatives, which provide us with the likelihood estimation of the success of an attack or defence **[2]**.

The third paper, "*A Machine Learning Based Two-Stage Wi-Fi Network Intrusion Detection System*" **[3]**, specifies how as the dependency on wireless networks grows, the number of network attacks grows as well. To counteract this, a machine learning implementation, a Wireless Network Intrusion Detection System (*WNIDS*) was created using the publicly available Aegean Wi-Fi Intrusion Dataset (*AWID*) **[3]**. The two-staged system that was implemented achieved an accuracy of 94.42% for multi-class classification with a reduced set of features **[3]**. This was lower than the other WIDS paper's accuracy of 98.8% **[1]**, which led us to not choose the third paper's approach for our implementation.

# Methodology

Our implementation is based on the research paper, "*WIDS: An Anomaly Based Intrusion Detection System for Wi-Fi (IEEE 802.11) Protocol*" **[1]**. In the paper, it states that a WIDS models the normal behaviour of data transmitted using a Wi-Fi protocol, by monitoring the state transitions of the Wi-fi protocol state machine. The received features, `frame_epoch_time`, `address_1`, `address_2`, `address_3`, `address_4`, `frame_type`, and `frame_subtype` **[1]**, are extracted for analysis from the raw Wi-fi frames, and the observed traffic between a source and destination is converted into a flow of data called an observation flow **[1]**. From this observational flow, n-grams of size 4 are extracted, where the frames within the n-grams follow one of these patterns of specific request types: (Auth, Auth, Asso, Data), (Auth, Asso, Data, Data), (Asso, Data, Data, Data), (Data, Data, Data, Deauth), or (Data, Data, Deauth, Deauth) **[1]**. An observation-wireless flow is a continuous flow of frames or packets between a source-destination pair over an interval of time *t* **[1]**. The paper determines the optimal time to be 10 seconds **[1]**. N-grams of any size can be used to model the temporal behaviour of the Wi-fi protocol. However, this paper determines the most effective n-gram size to be 4 frames **[1]**.

In the extraction process from a raw Wi-fi frame, the time at which the network card observed the frame, the addresses representing transmission sources and destinations, and information about the type of frame are extracted. In the paper, there are 4 different addresses in the Wi-fi frame **[1]**, however, in our model, we only take into account two of them, comprising the source and destination address of the frame.

For the data preprocessing, in order for the Wi-fi traffic to be transformed into an observation-wireless flow, the `frame_type` and `frame_subtype` are hashed together creating a combined field called a type **[1]**. This type is equivalent to the `wlan.fc.type_subtype` field, found within an 802.11 packet. The n-grams are formed using a sliding window over the type field, where the sliding window is equal to the size of the n-grams **[1]**. After the n-grams are formed, the frequency of each unique n-gram is calculated to get the probability of the flow **[1]**. This step is necessary for determining the optimal size of n-grams, however, it is not needed in our implementation as the results of the paper state n-grams of size 4 are optimal **[1]**.

In terms of training the models, the WIDS paper outlines 2 modules; Sniffer Module and Behavior Authentication Module (*BAM*) **[1]**. The Sniffer Module collects Wi-fi traffic using the configured network card set in monitor mode **[1]**. The Wi-fi traffic is then stored within a shared database that enables transfers and sharing of information with BAM **[1]**. BAM reads the raw Wi-fi traffic within the shared database and performs an analysis by converting the traffic into flows **[1]**. A machine learning model will then take the extracted n-grams and use them to classify the flows as either normal or abnormal **[1]**. The detector will output a 1 to classify the flow as abnormal, or a 0 to classify it as normal **[1]**. This process outlines how the paper trained their models, however, we use a custom database of AWID3 **[4]** entries to train ours instead of a database built from live captures.

# Development Technologies

Our development began with creating and training the Machine Learning model. We decided to develop our model in Python **[5]** due to various libraries that could be used for our implementation. Although we mentioned using the TensorFlow framework

**[6]** for the machine learning model in our Project Proposal, we eventually changed our approach and decided to use the Scikit-Learn library **[7]**. While we could have used TensorFlow, it is a much lower-level library that would have added unnecessary complexity to our algorithm whereas Scikit-learn is a higher-level library that includes various algorithms for classification such as AdaBoost and Random Forests **[7]**. As previously explained, the model used the extracted n-gram flows and classified them as normal or abnormal. More specifically, each Wi-Fi frame was classified as normal or abnormal, when frames are formed into n-grams, the presence of an abnormal frame marks the whole n-gram as abnormal. Being able to use these classifiers made Scikit-Learn a more attractive option as these were both classifiers the paper used to train and then analyze the performance of the model.

As we changed what library we were using for our machine learning model, some of the subsequent libraries we were planning on using had to change as well. Instead of using Keras **[8]** for reading in the datasets, Pandas **[9]** was used instead. As the paper mentioned using the AWID dataset for training and simulation, we also requested a copy of the datasets to limit any variables when doing comparisons from our implantation to theirs. While the WIDS paper used the AWID2 (*2016*) dataset **[10]**, we opted to use the newer AWID3 (*2021*) dataset **[4]** as it includes a greater number of modern Wi-Fi attacks. These datasets included Wi-Fi frames that could be used for training as well as testing against various attacks. As these datasets were in the form of CSV files, Pandas allowed us to read specific column names into a Pandas data frame, and identify various features observed from the wireless flows. Once data was read into the program, NumPy **[11]** was used to create n-dimensional arrays for data storage, giving us the ability to easily perform operations on the arrays, such as reshaping and indexing. Once the n-grams were made and the model was trained, Pickle **[12]** was used to serialize or "*pickle*" the model and save it to a file. To do predictions on the n-gram flows, the model is then "*unpickled*" and does predictions on the flows it is being fed.

In addition to testing the model on the data from the AWID3 dataset **[4]**, it was also tested on flows from live Wi-Fi capture. We started off by using Pyshark **[13]** but

ran into various problems such as frames getting cut off when capturing for 10 seconds, as well as getting Pyshark into monitor mode to capture 802.11 packets. We resolved these issues by switching to using tcpdump **[14]** on macOS. When doing live captures as well as working with the frames from the AWID3 dataset **[4]**, we used Wireshark to visualize and display the frames **[15]**. The frames being displayed in Wireshark allowed for the filtering of different frame types, as well as a much easier understanding of each frame and its features. This allowed us to see the differences between the live capture, the captures of normal data from the dataset, as well as the attack captures from the dataset that included the various Wi-Fi attacks. Lastly, we used Matplotlib **[16]** to analyze the output data produced by the testing of our models and produce the various graphs shown in this report.

# Algorithm Description

The models are generated, trained and tested in the `WIDSNetwork.py` file. When testing or training with data from a CSV file, it must be first parsed into a format that our implementation can read. In the `get_custom_data_from_dataset` function, relevant columns are loaded into a pandas data frame, and then written to a new CSV file.

Using the `get_n_grams_from_custom_dataset` function, we can extract the n-grams from the saved custom CSV file created in the previous step. This function calls the helper method `create_n_grams_from_dataset_features` from the `CaptureToFlow` class. This helper function extracts all n-grams of length 4 from a given dataset and also hashes the MAC addresses so they can be used by the models. The original function then classifies each n-gram as normal or abnormal depending on the labels of the frames within the n-gram.

The `train_network_from_classified_flows` functions train a model using a specific classifier, an array of n-grams of size 4 and the corresponding label representing the normalcy of the n-gram. Once the model is trained it is saved to a file using the pickle library. The `used_trained_model_to_predit_flow` function can then be used to load a model that has been saved to a file, and use it to predict the

normalcy of a given flow of n-grams. The predicted values returned by this function can then be used in several other functions to generate accuracy metrics.

The `generate_custom_attack_csv_files` function generates CSV datasets with information relevant to our implementation, for each attack type within the AWID3 attack database **[4]**. These CSV datasets are then transformed into n-grams, and fed into a given model to generate attack accuracy metrics.

Testing the model against live data can be done by either extracting n-grams from a live capture object or by saving the live capture to a PCAP file which would be processed using similar methods mentioned in the above steps. If using a live capture object, the helper method `extract_feature_set_from_live_capture` from the `CaptureToFlow` class is called to capture live packets in intervals of 10 seconds. This live capture object is then passed to the `create_n_grams_from_observed_features` helper function from the `CaptureToFlow` class, which extracts the n-grams. Finally, a given trained model is used to predict the normalcy of the live captures. The `generate_prediction_graphs` function is used to generate all of the graphs seen in this report after all accuracy metrics have been collected.

# Simulation

In this project we simulated attacks on a network using existing packet captures from the AWID3 attack dataset **[4]**. The simulated attack types were: Deauthentication attacks, Disassociation attacks, Re-Association attacks, Rogue Access Point attacks, Krack attacks, Kr00k attacks, SSH attacks, Botnet attacks, Malware attacks, SQL Injection attacks, SSDP attacks, Evil Twin attacks, and Website Spoofing attacks **[4]**. These attacks differ from the ones mentioned in the paper, as the paper uses an older version of the AWID dataset from 2016 **[10]**. Some of these attacks such as Kr00k were not discovered until 2019 and therefore were not included in the AWID dataset in 2016. The common attacks used by both our implementation and the WIDS paper **[1]** include Deauthentication attacks, Dissociation attacks, and Evil Twin attacks. Within the new attacks added to the AWID3 dataset **[4]**, there are some categories that represent multiple attacks. The Malware category represents the use of WannaCry and TeslaCrypt

ransomware, the Krack category represents man in the middle and key reinstallation attacks, and the Website Spoofing category represents ARP and DNS poisoning attacks **[4]**.

In order for our model to predict the normalcy of the wireless frames given for each attack. We created an observational flow of n-grams from each of the attack captures. Each n-gram flow was then fed into one of the trained models to get the accuracy metrics for each attack type, model combination. This project simulated the implementation of the wireless intrusion detection system using training data from the AWID3 dataset **[4]**. A combination of data from each attack and normal traffic was joined into a master dataset with roughly one million entries. This dataset was then broken down into flows of n-grams with size 4 as this was the optimal size determined in the paper, which resulted in a total of 9821 n-grams. These n-grams were split into training and tests sets, 5000 were used in the training set which was used to train the models, and the remaining 4821 n-grams were used to determine the general test accuracy of the trained models.

Our model implementations use two different supervised learning algorithms, each model is trained on the same data, but with a different classification algorithm. The first model uses an AdaBoost classifier **[17]**, which belongs to the category of ensemble machine learning. This classifier groups together multiple classifiers to increase accuracy. The initial model for an AdaBoost classifier is trained using the training dataset and one of the sub-classifiers, this produces a set of predictions that are used to update weights. The updated weights are then used with another one of the sub-classifiers, to update the machine learning model. This process iterates until we hit the maximum number of estimators, which we defined as 100. The second model uses a RandomForest classifier **[18]**, which also belongs to the category of ensemble machine learning. However, this classifier works by creating a number of separate decision tree classifiers for various samples in the dataset. The number of decision tree classifiers to create is equal to the number of estimators, which we defined as 100. Within these decision trees, splitting a node is performed by searching for the best feature in a random subset of features. This chosen feature may not be the feature with

the overall highest activation but allows for a more diverse model, as a variety of features can be included in each decision tree. At the end of the RandomForest classification algorithm, the decision trees are merged together to create the final model.

# Results and Analysis

In order to determine which classification algorithm resulted in the best machine learning model, the general test accuracy of the two models were compared. The AdaBoost classifier **[17]** achieved a general test accuracy of 97.14% and the RandomForest classifier **[18]** achieved a general test accuracy of 96.8%. Both accuracies achieved were extremely close to the 98.8% accuracy outlined in the WIDS paper **[1]**. Based on these results, AdaBoost was used as the classifier for our primary model which was used to predict the normalcy of live traffic.

When testing the accuracy of our trained primary model against live capture, it was 100% accurate in determining normal activity. Live capture was converted into observational flows in intervals of 10 seconds, which were then broken down into n-grams of size 4 and fed into the trained primary model to make a prediction. We did not manage to capture live wireless attacks, so the accuracy of the primary model in predicting abnormal activity was determined using captures for specific attacks from the AWID3 attack dataset **[4]**.

Below are visual representations of the results collected in our implementation, as well as comparisons against the results achieved in the WIDS paper **[1]**. These results include the metrics of accuracy, detection time, extraction time, true positives, and false positives. True positives represent instances where the models' prediction and the actual value are both abnormal. False positives represent instances where the models predict abnormal activity when the actual value of the data is normal. Detection time represents the time our primary model takes to determine the normalcy of a flow of n-grams, and extraction time represents how long it takes to extract the n-grams from the observed frames.

The detection time for our primary model is roughly 0.071 seconds, meaning that the model is able to almost instantly predict the normalcy of captured n-grams. However, the process of extracting n-grams from observed frames is a more expensive operation and takes roughly 47.6 seconds. These metrics of detection and extraction time were not mentioned in the WIDS paper **[1]**, and thus cannot be compared.

The first table below, **Table.1**, shows the accuracy of the models using the AdaBoost **[17]** and RandomForest **[18]** classifiers, in predicting the attacks within the AWID3 attack dataset **[4]**. **Table.2** and **Table.3** compare our obtained true and false positive percentages against the results from the WIDS paper **[1]**. Lastly, **Figure.1** and **Figure.2** show visual representations of the accuracy metrics obtained in our implementation.

| ***Accuracy Per Attack and Classifier*** | **AdaBoost** | **RandomForest** |
|---|---|---|
| **Re-Association Attack** | 68.3% | 0% |
| **Botnet Attack** | 0% | 100% |
| **Deauthentication Attack** | 99.85% | 83.75% |
| **Dissociation Attack** | 87.5% | 87.5% |
| **Evil Twin Attack** | 0% | 0% |
| **Kr00k Attack** | 80% | 100% |
| **Krack Attack** | 57.14% | 14.3% |
| **Malware Attack** | 72.72% | 100% |
| **RougeAP Attack** | 80% | 100% |
| **SQL Attack** | 76.92% | 100% |
| **SSDP Attack** | 0% | 100% |
| **SSH Attack** | 72.72% | 100% |
| **Website Spoof Attack** | 72.72% | 100% |

*Table 1: Classifier accuracy for predicting different attack types*

| | Deauthentication | | Dissociation | | Evil Twin | |
|---|---|---|---|---|---|---|
| | True Positve | False Positive | True Positve | False Positive | True Positve | False Positive |
| **AdaBoost (Ours)** | 99.85% | 0.15% | 81.25% | 12.5% | 0% | 100% |
| **AdaBoost (Paper)** | 93% | 7% | 100% | 0% | 0% | 100% |

*Table 2: True and false positive comparison for AdaBoost classifier models*

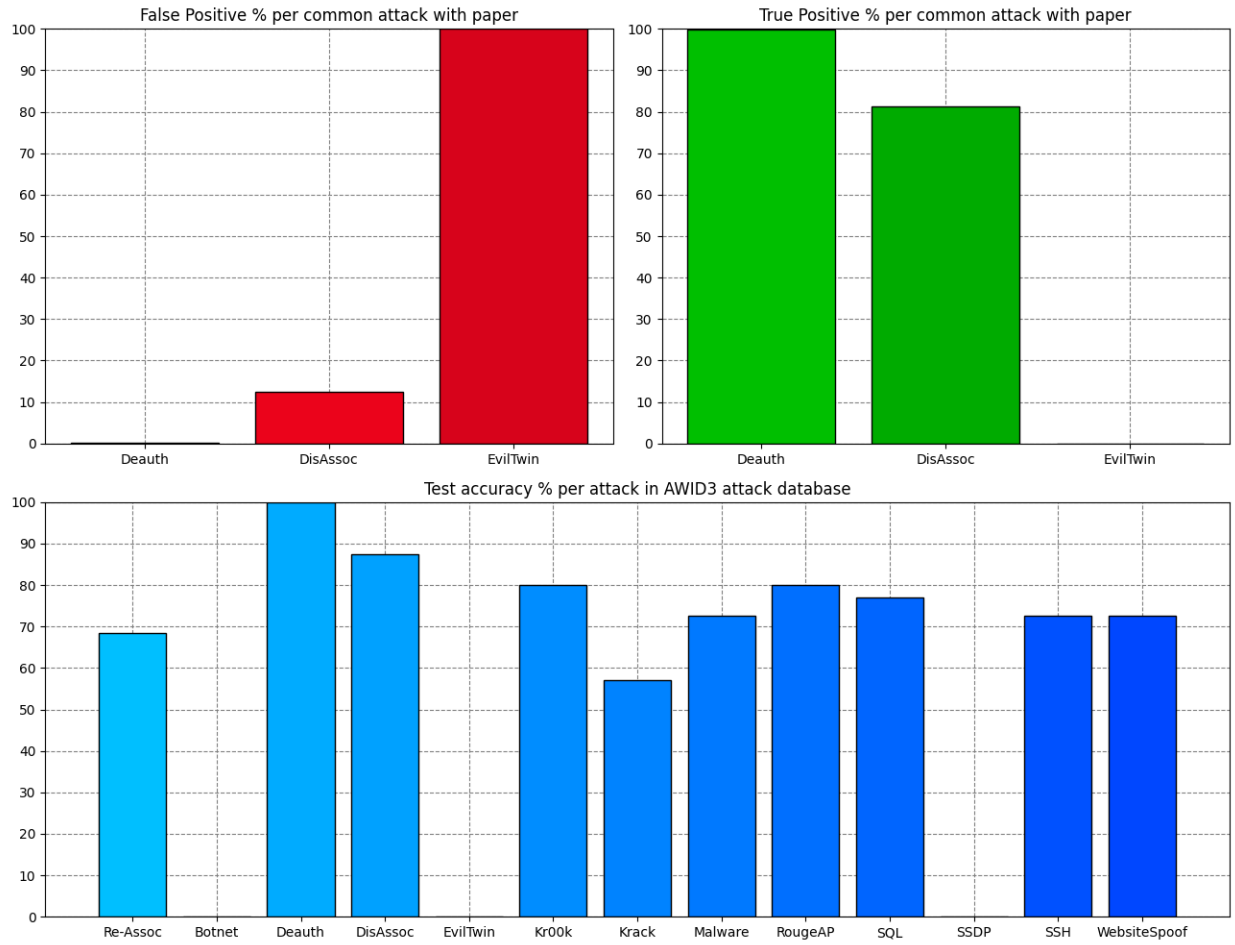


*Figure 1: Accuracy graphs for the AdaBoost classifier model*

| | Deauthentication | | Dissociation | | Evil Twin | |
|---|---|---|---|---|---|---|
| | True Positve | False Positive | True Positve | False Positive | True Positve | False Positive |
| **Random Forest (Ours)** | 83.6% | 0% | 68.75% | 0% | 0% | 100% |
| **Random Forest (Paper)** | 95% | 5% | 98% | 2% | 0% | 0% |

*Table 3: True and false positive comparison for RandomForest classifier models*

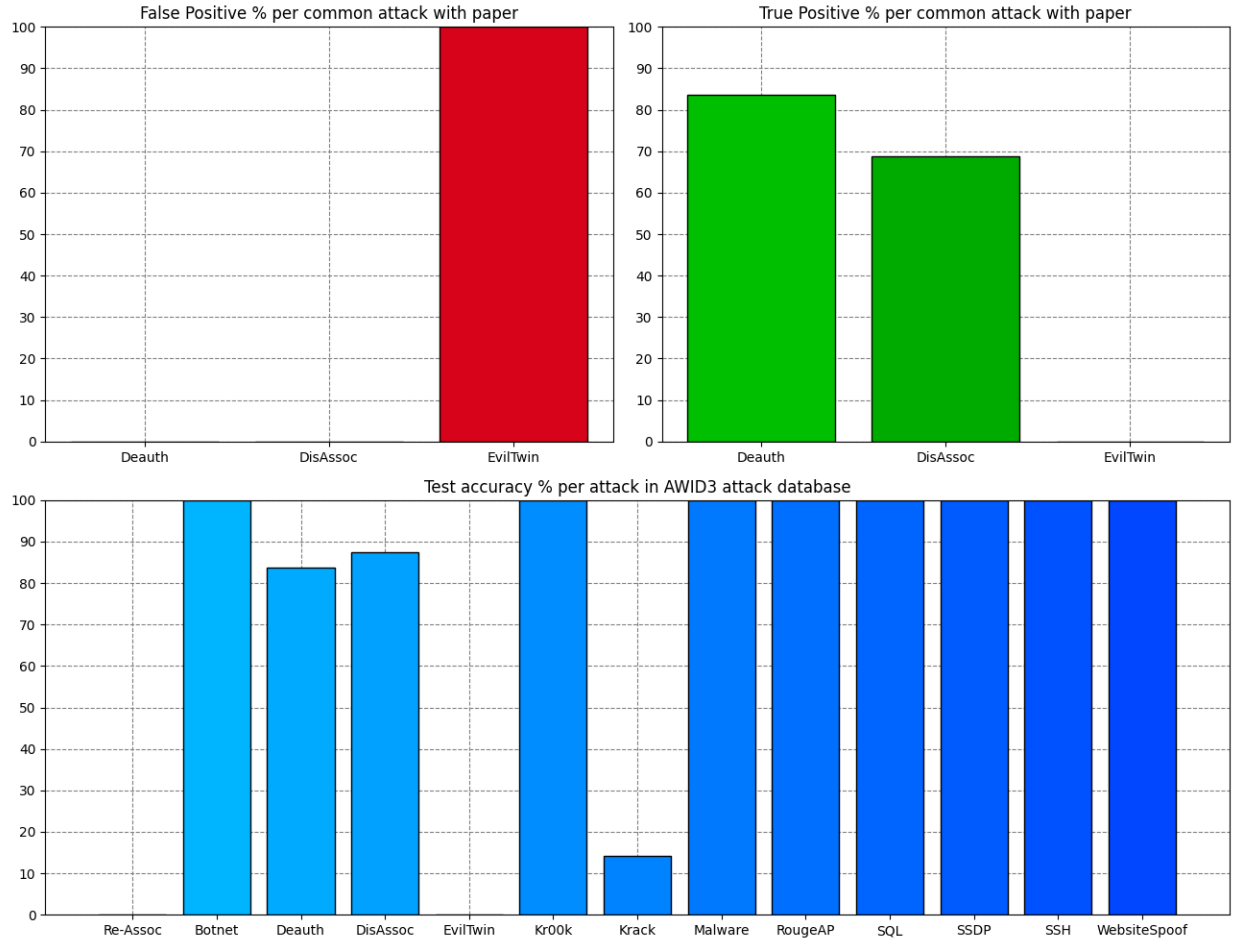


*Figure 2: Accuracy graphs for the RandomForest classifier model*

As shown in **Table.2**, our AdaBoost classifier **[17]** model performed better than the WIDS paper **[1]** in detecting deauthentication attacks. Our implementation achieved a true positive rate of 99.85% and a false positive rate of 0.15%, whereas the WIDS paper achieved a true positive rate of 93% and a false positive rate of 7% **[1]**. With regards to dissociation attacks, our implementation performed slightly worse than the WIDS paper **[1]**, with a true positive rate of 81.25% and a false positive rate of 12.5%, compared to the 100% true positive rate and 0% false positive rate achieved in the WIDS paper **[1]**. These differences could be due to a variety of factors, however, they most likely stem from the difference in training and test data between the AWID2 **[10]** and AWID3 **[4]** datasets. Evil twin attacks were unable to be classified by both our implementation and the WIDS papers implementation **[1]**. This is because both models are unable to distinguish the access point used in an evil twin attack as illegitimate, as the access point created by the attack behaves the same as any legitimate access point **[1]**.

As shown in **Table.3**, our RandomForest classifier **[18]** model performed worse than the WIDS paper in detecting all common attacks **[1]**. Although our implementation has a 0% false positive rate for deauthentication and dissociation attacks, which is better than the respective 5% and 2% rates achieved in the WIDS paper **[1]**. The true positive rate for deauthentication and dissociation is only 83.6% and 68.75% respectively, whereas, the WIDS paper achieves a 95% rate for deauthentication attacks and a 98% rate for dissociation attacks **[1]**. Our implementation also has a 100% false positive rate for evil twin attacks, meaning that each one is incorrectly predicted. However, the WIDS paper **[1]** has a 0% true and false positive rate for evil twin attacks, meaning that the attacks are not recognized at all. The poor performance of this model compared to the model from the WIDS paper **[1]** further solidified our decision to use the AdaBoost classifier **[17]** model as the primary model for our implementation.

One benefit of using the model trained with the RandomForest classifier **[18]** can be seen when looking at the general test accuracy per attack in **Figure.1**, **Figure.2**, and **Table.1**. Our RandomForest classifier **[18]** model was able to predict a larger variety of

attacks than our AdaBoost classifier **[17]** model. The RandomForest classifier **[18]** model can predict both botnet and SSDP attacks, whereas the AdaBoost classifier **[17]** model could not predict either. However, the AdaBoost classifier **[17]** model is able to predict re-association attacks, whereas the RandomForest classifier **[18]** model is not able to.

In summary of the results mentioned above, the model trained using an AdaBoost classifier **[17]** was the most accurate at detecting abnormal network traffic, and performed similarly to the WIDS paper's implementation **[1]**. However, even though the RandomForest classifier **[18]** implementation performed worse than the WIDS paper's **[1]**, it was able to detect new attack types within the AWID3 dataset **[4]** that were not included in the previous AWID2 dataset **[10]**.

# Conclusion

With new types of wireless network attacks being created every year, securing wireless networks is an ongoing problem. Researchers are continuously looking for new and innovative ways to protect networks. We chose to create an implementation based on the WIDS paper **[1]** that accomplishes the goal of intrusion detection using machine learning models. This paper was chosen as it was able to detect abnormal wireless activity with a 98.8% accuracy **[1]**. Given the original goal of implementing a system for the real-time detection of network intrusions, our project provided promising results, similar to the results achieved within the WIDS paper **[1]**. Our implementation based on the WIDS paper **[1]** is capable of detecting a greater number of attack types, including Botnet, Kr00k, Krack, Malware, Rogue AP, SQL injection, SSDP, SSH, and Website Spoofing attacks. In closing, this project provides a new solution to the problem of wireless intrusion detection, by delivering trained machine learning models capable of detecting abnormal networking activity with relatively high accuracy, and low false-positive rates.

# References


**[1]** P. Satam & S. Hariri, 'WIDS: An Anomaly Based Intrusion Detection System for Wi-Fi (IEEE 802.11) Protocol', *IEEE Transactions on Network and Service Management*, 18(1), 1077–1091, 2021. https://doi.org/10.1109/TNSM.2020.3036138

**[2]** J. F. M. Sánchez, O. J. S. Parra, & J. Medina, 'Design and Implementation of an Intrusion Prevention System for Wi-Fi Networks 802.11 AC', Cham: Springer International Publishing, 2019, 32–41. https://doi.org/10.1007/978-3-030-03101-5_4

**[3]** A. A. Reyes, F. D. Vaca, G. A. Castro Aguayo, Q. Niyaz, & V. Devabhaktuni, 'A Machine Learning Based Two-Stage Wi-Fi Network Intrusion Detection System', *Electronics*, 9(10), 2020. https://doi.org/10.3390/electronics9101689

**[4]** E. Chatzoglou, G. Kambourakis, & C. Kolias, 'Empirical evaluation of attacks against IEEE 802.11 enterprise networks: The AWID3 dataset', *IEEE Access*, 9, 34188–34205, 2021. https://doi.org/10.1109/ACCESS.2021.3061609

**[5]** Python Software Foundation. Python Language Reference, version 3.8. (Online) Available: http://www.python.org

**[6]** Martín Abadi, et al., 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems'. 2015. (Online) Available: https://www.tensorflow.org/

**[7]** F. Pedregosa, et al., 'Scikit-learn: Machine Learning in Python', *Journal of Machine Learning Research*, 12, 2825–2830, 2011.

**[8]** F. Chollet & Others, 'Keras', 2015. (Online) Available: https://keras.io

**[9]** T. P. D. Team, *pandas-dev/pandas: Pandas*. Zenodo, 2020. https://doi.org/10.5281/zenodo.3509134

**[10]** C. Kolias, G. Kambourakis, A. Stavrou, & S. Gritzalis, 'Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset', *IEEE Communications Surveys & Tutorials*, 18(1), 184–208, 2016. http://doi.org/10.1109/COMST.2015.2402161

**[11]** C. R. Harris, et al., 'Array programming with NumPy', *Nature*, 585(7825), 357–362, 2020. https://doi.org/10.1038/s41586-020-2649-2

**[12]** G. Van Rossum, *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020. (Online) Available: https://docs.python.org/3/library/pickle.html

**[13]** Dor Green, pyshark, GitHub repository, 2022. (Online) Available: https://github.com/KimiNewt/pyshark

**[14]** The TCP Dump Group, tcpdump, Github repository, 2022. (Online) Available: https://github.com/the-tcpdump-group/tcpdump

**[15]** Wireshark, 2022. (Online) Available: https://www.wireshark.org/

**[16]** J. D. Hunter, 'Matplotlib: A 2D graphics environment', *Computing in Science & Engineering*, 9(3), 90–95, 2007. https://doi.org/10.1109/MCSE.2007.55

**[17]** Scikit-Learn, Scikit-Learn API Reference, AdaBoostClassifier. (Online) Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

**[18]** Scikit-Learn, Scikit-Learn API Reference, RandomForestClassifier. (Online) Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html