Capable of Classifying the Tuples with Wireless Attacks Detection Using Machine Learning

Conference Paper · March 2023

DOI: 10.1007/978-3-030-98457-1_1

2 authors:

Tariqul Islam
Daffodil International University
20 PUBLICATIONS **16** CITATIONS
SEE PROFILE

Shaikh Muhammad Allayear
Daffodil International University
54 PUBLICATIONS **365** CITATIONS
SEE PROFILE

Capable of classifying the tuples with wireless attacks detection using Machine Learning

Tariqul Islam[1], Dr. Shaikh Muhammad Allayear[2]
[1, 2]Daffodil International University, Dhaka, Bangladesh

[1]tariqul15-2250@diu.edu.bd, [2]headmct@daffodilvarsity.edu.bd

Abstract. A wireless attack is a malicious action against wireless systems and wireless networks. In the last decade of years, wireless attacks are increasing day by day and it is now a very big problem for modern wireless communication systems. In this paper, the author's used the Aegean Wi-Fi Intrusion Dataset (AWID3). There are two versions of this dataset, one is over 200 million tuples of full data and one is 1.8 million tuples of reduced data. As our dataset has millions of tuples and over 100 columns, it is easy to become overwhelmed because of its size. The authors used the reduced version and predicted if an attack was one of four types using the k-nearest-neighbors classifier. All of the attack types we used had a distribution that highly favored the non-attack class. Our best results were for the attack "arp" type where we attained the best accuracy with recall. The author's primary goal of the paper was to attain the highest accuracy possible when creating a model that is capable of classifying the 4 attack types and detecting and classifying wireless attacks using Machine Learning models on the AWID3 dataset. One of the goals that supported this main objective was determining a way to avoid the curse of dimensionality.

Keywords: Data Mining, AWID3, Cybersecurity, Machine Learning, Wireless Security

1 Introduction

Wireless attacks have become a very common security issue. Different kinds of attack methods can happen such as "Rogue access points, Jamming or Interference, Evil twin, Wardriving, Bluejacking, Packet sniffing, Near field communication, WEP or WPA attacks, WPS attacks, IV attack, Warchalking".

A rogue access point is an unauthorized access point that can be added to one's wireless network without one's knowledge. Being able to combat this can be done by having network access controls in place or by walking around one's building. The evil twin is a technique used by criminals to gain unauthorized access to a network. With this method, they only need to purchase a wireless access point and then configure it as exactly as the network it's connected to. Doing so allows them to monitor and gain control of all the details of the wireless network. Bluejacking is an illegal practice that enables people to send and receive messages to another device through Bluetooth. This

is similar to hacking, where the goal is to send an email or a message to another device. Bluesnarfing is a far more dangerous type of attack that involves taking advantage of a vulnerable Bluetooth network to steal sensitive information. This vulnerability can allow an attacker to access a mobile device's contacts and images. War chalking is a method used to determine the location of a wireless access point. An Initialization Vector attack is a type of wireless network attack that can cause modification to the Initialization Vector of a packet. Analyzing the details of a given packet after an attack is performed will allow an attacker to extract valuable information from it. Packet sniffing is a very challenging technique when it comes to wireless networks. In this process, an individual can capture a packet that is sent across a network and see the details of the message that it contains. This feature makes it incredibly easy for an individual to read and hear what's happening in real-time. This technique allows one device to collect data from another device that is in range. An attacker can also collect details about a conversation between two people. An attacker with this kind of information can easily gain access to a computer without its owner's knowledge. He can also use it to prove his identity or authenticate himself. In some cases, a legacy wireless access point can be considered an encryption method that is not suitable for sensitive data. Attacks with the use of WPS passwords are very dangerous and can lead to unauthorized access to a particular network. These types of attacks are usually carried out by criminals who are looking to steal sensitive information. With the help of this tool, an attacker can easily obtain the passwords of wireless networks and gain access to the data and information that is on their network.

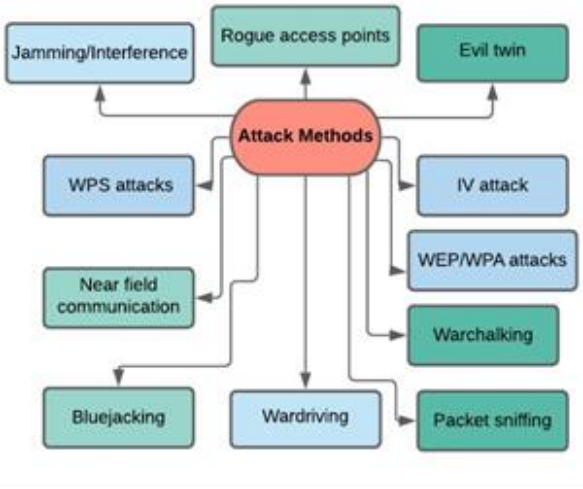

Figure 1: Different attack methods

2 Related work

In the paper [1] implementation of AMBA-AXI protocol by using VHDL for SoC (System on a Chip) integration with different components and IPs. But the main challenge in the paper is to verify chip communication properties. They propose a protocol checker such as rule-based AMBA AXI and this contains almost 44 different rules for the check. Bisht et al. [2] develop the first systematic approach for detecting parameter tampering opportunities in web applications and implement this approach in a tool that is called NOTAMPER. This paper presents an approach that combines JavaScript code analysis and black-box vulnerability analysis. It tackles many of the issues traditional methods face in generating and validating inputs. Although it eliminates many of the issues faced by traditional methods, it still requires some effort and compromises. Soleimani et al. [3] research work proposes a black-box method to detect web application vulnerabilities including XSS, CSRF, and information leakage. Takamatsu et al. [4] propose a technique that automatically detects session management vulnerabilities in web applications by simulating real attacks. This method requires the test operator to enter a few details about the web application before testing it. However, it can detect the existence of special characters on the web page. This method can only detect the existence of the special string. Kumar et al. [5] research work propose a detection mechanism of session fixation vulnerability. This paper has limitations like the automated solution architecture of session fixation vulnerability detection has not been defined using any prototype. The model elements are session Id, prepare post request using the valid credential, capture Authenticated session, match authenticated and unauthenticated session. Nadar et al. [6] propose a model that can detect attacks such as cross-site request forgery attacks, broken authentication. Model elements are CSRF: "Request Checker, Packet Tracer, Request Differentiator, Attack Handler, Module, Notify Client, Attack Database Repository". Important result parameters are "True positive, True negative, False positive, False negative, accuracy, Recall, Precision". Dahse and Holz's [7] research paper, they are the first to propose an automated approach to statically analyze second-order data flows through databases, file names, and session variables using string analysis. They propose to detect multi-step and second-order exploitation risks in web applications. They study the issue of second-order sanitization. We discovered 159 previously unknown security issues in the code of our clients. Some of these issues were caused by exploitation techniques such as XSS and remote code execution attacks. Three false negatives were reported in Scarf due to the parser not handling SQL string functions. This vulnerability was discovered in the following list of attacks. Aliero et al. [8] proposed solution is the only solution that performs stored procedure attacks SQL and bypass login authentication even if the returned records are limited restriction is applied. The following steps are needed to trigger SQL injection vulnerability and store procedure vulnerability in web pages. These steps will introduce the necessary vectors and techniques to trigger SQL injection attacks. Ze et el. [9] proposed an efficient Web application vulnerability detection mechanism based on optimized crawler and feature recognition. This research introduces black-box testing ideas and fuzzy testing technology into Web vulnerability detection and realizes a Web vulnerability detection system based on page code analysis and page code analysis modules. Jovanovic

et al [10] proposed flow-sensitive, interprocedural, and context-sensitive data flow analysis. The complexity of the PHP analysis process was due to its untyped nature. Additional steps such as alias analysis and literary analysis lead to more precise and comprehensive results. The proposed system, called Pixy, was developed to address these issues. Marashdih and Zaaba at el. [11] shows detection of XSS vulnerability, removing the detected vulnerability. But didn't consider an experiment on DOM-based XSS. Medeiros and Neves at el. [12] propose the insights from the behaviors of SATs, they analyze applications written. But the research work is done with only one vulnerability which is SQLi. Lukanta et al. [13] propose to detect session management vulnerabilities, we developed a vulnerability scanning tool extending an existing open-source tool, namely Nikto. But the random token detection mechanism should be revised due to errors in random token detection. Yuan et al. [14] investigate vulnerabilities and analyze the common vulnerabilities in web applications. Anbiya et al [15] in the paper will present an approach using static analysis with lexical analysis techniques for reducing an expert's knowledge. Lexical analysis is a static analysis technique that transforms source code to other representations, usual tokens for future use. But this research only works with SQL injection, cross-site scripting, and directory traversal. Jeevitha and Bhuvaneswari et al. [16] paper-primarily focus on detecting the malicious node. But some limitations in this paper such as prevention techniques for the malicious nodes are not included in this paper. Pan and Mao. et al. [17] propose the notion of DOM-sourced XSS which emphasizes the hierarchical document source and distinguishes it from DOM-based XSS. Mokbal et al. [18] propose extensive real-world data composed of 138,569 unique records to detect XSS attacks that have been constructed comprehensively and uniquely. Kao et al. [19] review different types of SQLi attacks, provide their descriptions, and analyze possible investigative techniques. Moustafa et al. [20] proposed a methodology to automatically capture web data, network traffic data for extracting relevant features. Alswailem et al. [21] feature a combination that comes from the Random Forest technique, as it has high accuracy, is relatively robust and has a good performance. Wang et al. [22] summarize the framework of access control vulnerability detection. Deepa et al. [23] proposed a system called "XiParam"; this prototype is not tested in real-world web applications due to its dependence on the database driver files. Figueiredo et al. [25] developed a learning-based tool that can detect input validation vulnerability using static data flow analysis. But the study did not compare their outcome with another dictation tool. Ibarra-Fiallos et al. [26] design a durable, reliable, and effective protection filter for reducing common web injection attacks. The evolution of web applications, as well as the need to protect them against injection attacks, requires the development of a filter that can safeguard the various parameters of web applications. Choi et al. [27] used static and dynamic analysis approaches, where Static analysis is used to boost up XSS vulnerability detection speed and automatically extracts from a web page and filters out duplicates which saves time.

3 Research Methodology

Here the author's used three different programming languages to assist us: Python, R-Studio, and Microsoft SQL. For Python, the author's used the latest version available at the time of writing Python 3.9.0. In this language, we used the Scikit-Learn, Pandas, and NumPy libraries to help us with our preprocessing steps. Then, in R we used the DMwR library for its SMOTE and K-nearest-neighbors functionality. Finally, we used Server Management Studio to perform queries for Microsoft SQL. We imported the reduced version of our dataset into it to use the SQL server after adding the headers to it. Then reduced AWID3 dataset [28] with only the columns posted on the class web page in Microsoft SQL Server. Lastly, install a streamlet and other dependencies on the system. Ensure both the dataset and the python file are in the same directory. The details about how we were able to achieve this task are authentication discussed in greater detail in our preprocessing and data cleaning section. In the reduced version of the AWID3 dataset, there is a class variable with the values "normal, arp, cafe_latte, amok, deauthentication, authentication_request, beacon, evil_twin, fragmentation, and probe_response". The type normal represents the packets that are not an attack. The other types listed represent an attack. For the author's analysis, chose to use the k-nearest-neighbors classifier algorithm to predict if a packet was of the type: "arp, deauthentication, amok, or authentication_request".

An IP address spoofing attack is another type of attack that uses the address resolution protocol to trick a host into sending traffic to an attacker [29]. A deauthentication attacker uses unprotected deactivation packets. The attacker monitors network traffic to find out which MAC addresses are associated with certain customers. Then, a deauthentication message is sent to the access point on behalf of those particular MAC addresses to force a client of the network. Our third attack type is an authentication request attack. This is a flooding attack in which the attacker tries to exhaust the access point's resources by overflowing their client association table. The last type of attack is an amok attack. Though information on this particular form of cyberattack is limited, we determined that it is another flooding attack on a server, similar to the authentication request. This attack causes the client association table to overflow due to the number of clients that can be maintained there. An entry is inserted into the client association table after an Authentication Request message.

3.1 Processing method

Build a classifier capable of properly classifying tuples with four specific attack types: Amok, Deauthentication, Authentication Request, ARP. The authors had three major tasks during this work: preprocessing & data cleaning, feature selection, and classification.

3.2 Preprocessing & Data Cleaning

As previously mentioned, we used Python 3.9.0 to perform the majority of our preprocessing and data cleaning on our dataset. “The dataset focuses on WPA2 Enterprise, 802.11w, and Wi-Fi 5. It contains multi-layer and modern attacks, including Krack and Kr00k. It is available in pcap format” [30]. AWID3 (Aegean Wi-fi Intrusion) Dataset produced from real Wireless Network logging. This full dataset is shown in figure 2 and the reduced training set is shown in figure 3 that produced from 1 hour of logging. The majority of data is of the "normal" class in either dataset. To do this, we first downloaded the reduced dataset from the dataset’s host. Then, by using the read_csv method from the panda’s library, we were able to read in a subset of columns that we wanted to work with for our analysis. These 19 desired columns were the following: 2, 5, 45, 62, 64, 65, 68, 71, 74, 75, 88, 91, 92, 105, 106, 110, 116, 120, and 154. After reading in these specific columns, we iterated over a text file containing the attribute names for the columns in our dataset. If the index of our desired columns existed in that list of attribute names, we added it to a list. Then, we were easily able to set the column headers for our minimized dataset. After we had successfully removed unwanted columns from our larger dataset and given each column a name, we were ready to proceed with the rest of our preprocessing. To start this process, we replaced the question marks in the dataset with a NumPy NaN value. The authors performed this operation to enable us to use other methods provided in the panda library to remove missing values from our data set. After replacing these values, we iterated over the columns in our dataset and removed a column if over 60% of its values were NaN (Not A Number). If a large majority of a column in the dataset was NaN, then it was not going to be useful for our later analysis. By performing this step, we removed 7 columns from our already minimized dataset. Then we removed columns that had zero or one unique value. A column had zero unique values if there were no values provided, but was not filled in with a NaN value. In contrast, a column would have one unique value if it had the same value for every tuple. This target data would not contribute to our future analysis in both of these situations because there was nothing to help distinguish a tuple with a normal class value or one of our four attack types. As a final step in preprocessing and cleaning our dataset, we removed the rows with a target of at least one NaN value left. By doing this, we removed 1972 rows from the dataset. In hindsight, more caution should have been exercised when doing this as these tuples with a NaN value could have helped us further identify anomalies in the dataset. Last, after performing all these steps, we exported the reduced dataset as a CSV file for later use in our analysis. All of the code for performing the operations described in this section are shown in data_preprocessing_cleaning.py.

AWID-ATK-F-Trn		AWID-ATK-R-Trn	
12416	amok	31180	amok
1529284	arp	64609	arp
93011	authentication_request	3500	authentication_request
170826	beacon	1799	beacon
1860780	cafe_latte	45889	cafe_latte
817954	deauthentication	10447	deauthentication
23598	evil_twin	2633	evil_twin
1089	fragmentation	770	fragmentation
157749037	normal	1633190	normal
117252	probe_response	1558	probe_response

Figure 2: AWID3 Full dataset to Reduced Training Dataset (~1.8 Million Rows x 155 Columns)

Dropped columns not listed on the course webpage. Replaced '?' with NaN values, then dropped columns with over 60% NaN values and removed 7 columns. Drop the rows with at least one NaN value in it ~ 2000 rows. Output the relatively clean data to a new file. Perform min-max normalization on attributes used for classification (range 0-1).

1	0.0	0.000000e+00	0.00310559	normal
2	0.0	6.929365e-02	0.00310559	normal
3	0.0	4.656501e-03	0.00310559	normal
4	0.0	1.579527e-01	0.00310559	normal
5	1.0	1.184824e-03	0.64285714	normal
6	0.5	1.427499e-05	0.00310559	normal
7	0.0	4.765562e-02	0.00310559	normal
8	0.5	4.054096e-04	0.00310559	normal
9	0.0	8.013122e-02	0.00310559	normal
10	0.0	5.141851e-03	0.00310559	normal
11	0.0	3.567034e-02	0.00310559	normal
12	1.0	6.566494e-05	0.64285714	normal
13	0.5	1.084899e-04	0.00310559	normal
14	0.0	1.219912e-01	0.00310559	normal
15	0.0	7.223715e-02	0.00310559	normal
16	1.0	1.800932e-02	0.64285714	normal
17	0.5	1.427499e-05	0.00310559	normal
18	1.0	6.480844e-03	0.64285714	normal

Figure 3: Normalization Output

3.3 Feature Selection

To select which columns we would use with the classifier we first removed all columns except the columns listed on the course webpage. We then went through them by hand looking for attacks that had null values (? 's in our case) and non-null values for the normal type or vise-versa. None were found so we went on to the next step where we removed many null valued columns and rows (see the preprocessing/data cleaning section). Next, we used the following SQL query: Select Count(Distinct(<COLUMN_NAME>)) from <AWID_DATABASE> where class='<CLASSTYPE>' where <CLASSTYPE> equaled the normal and all attack classes we used and <COLUMN_NAME> was varied to equal the remaining columns in our data. Before deciding that a column wasn't going to be removed we looked at the data further to make sure that even if all classes had the same number of distinct values that they weren't different distinct values for the attack and the normal type. Below are the results of these queries and our analysis of them. We didn't choose the radiotap_flags_shortgi column because it had no variation. We chose the wlan_fc_type column because after looking at it further it showed that the normal class type varied highly among 3 values while the attack types all had the same values and this frame_time_delta_displayed variable because upon looking further into it we saw that for the attack types all the tuples had very close values while the normal type was more spread out. We didn't choose the wlan_fc_version, wlan_fcs_good, frame_offset_shift, frame_offset_shift and radiotap_rxflags_badplcp variable because of the lack of variation. And also the wlan_duration variable because there is a lot of variation for type normal and barely any for those whose class value is one of the four attack types.

After much debate, we decided to not use this variable (wlan_ra) even though it has the variation we are looking for because it is nominal and not ordinal and the rest of our variables are continuous. Since SMOTE only works for continuous variables we felt it would be best not to use it otherwise we would probably have to one-hot-encode all of our data and use a different classifier than K-Nearest-Neighbors and not use SMOTE. We didn't choose this attribute (wlan_fc_moredata) because even though it has the variation we were looking for, 99.99% of the normal tuples had the same value as the attack types making it useless. The authors attempted PCA but ran out of memory even on CSU's Big Data Servers. We examined distinct values in the remaining columns and chose those with more distinct values for the normal class value than the attack class values using a little SQL magic. Then, chose the following 3 columns for our analysis and Isolated the attack types. Separate files to handle each attack type are shown in figure 4.

Figure 4: Handle Each Attack

Partitioned dataset into 66.6% training data and 33.3% test data. Performed SMOTE on training data to create synthetic tuples of attack types. K-Nearest Neighbor classifier to train the model for each specific attack type. Made predictions using the model on the test dataset. Parameter Selection/Interpretation Recall - "completeness – what % of positive tuples did the classifier label as positive?" Recall = TP/TP+FN and Precision - "exactness – what % of tuples that the classifier labeled as positive are positive" that means Precision = TP/TP+FP. Recall and precision are inversely related measures, meaning as precision increases, recall decreases. Accuracy and recall are inversely related in our case (for a majority of our data).

3.4 Classification

Our classification code is written in R. We chose this because it was already known by the group member that wrote the classification code. To perform classification we decided early that we would use SMOTE to create synthetic minority class tuples because the dataset is highly biased towards the normal class. We decided on using the K-nearest-neighbor classification algorithm later. We chose the k-nearest-neighbor algorithm because it works well with continuous data and every variable we used to classify was continuous. To perform the classification we first min-max normalized all of our target attributes to [0,1]. Next, we removed all classes except the normal and the attack class that we were currently predicting. Then, we randomly divided the normalized dataset into 66.6% training and 33.3% test datasets. Next, we fed the training data into the function SMOTE() in R studio. Then, we fed the oversampled resulting training dataset from SMOTE and the test dataset into the function kNN() in R. From the kNN() function we received a vector containing a 0 or 1 depending upon if the attack class was predicted. We used different operands for the SMOTE() and kNN() functions depending upon if we wanted higher recall or higher accuracy. We attempted cross-validation but due to an error we received ("too many ties"), we were not able to complete it.

4 Result & Discussion

We performed multiple tests on each attack type and recorded our top two, except for ARP. Our results for ARP were extremely good, which made it very challenging to further improve them. We either ran into errors or achieved similar results each time we attempted to modify the parameters. By tuning the parameters for SMOTE and the KNN classifier we were able to improve our accuracy and precision for each of the other attacks, but at the cost of the recall. Several tables displaying the results and parameters used for each of the four attacks can be seen below. We are quite certain the results for ARP were significantly better than the other three attacks because some of the variables for ARP had mutually exclusive values, while the same variables for the other attack type shared values with the normal type. For example, for ARP the fc_type variable was always 0 and the wlan_duration variable for ARP was always the same while that was not the case for the other attacks. Performed multiple tests for each

attack. ARP (Test 1) KNN Parameters - Smote.k = 3, KNN .k = 5, smote.perc.over = 150, smote.perc.under = 90 Confusion Matrix- N = 576,582.

Table 1. ARP (Address Resolution Protocol) Test 1

	Predicted: NO	Predicted: YES	Total
Actual: NO	552,958	1,731	554,689
Actual: YES	4	21,889	21,893
Total	552,962	23,620	

Table 2. ARP (Test 1) - Anomaly Detection Metrics

False Positives	1,731
True Positives	21,889
True Negatives	552,958
False Negatives	4

Table 3. ARP (Test 1) - Anomaly Detection Metrics (Contd.)

Accuracy	99.6990%
Error Rate	0.3009%
Sensitivity	92.6714%
Specificity	99.9992%
Precision	92.6714%
Recall	99.9817%

Only one set of results with ARP. Difficult to improve on already extremely good results for ARP. Amok (Test 1) KNN Parameters- Smote.k = 3, knn.k = 5, smote.perc.over = 150, smote.perc.under = 90. Amok (Test 1) - Confusion Matrix- N = 565,216

Table 4. Amok (Test 1)

	Predicted: NO	Predicted: YES	Total
Actual: NO	511,451	42,928	554,379
Actual: YES	562	10,275	10,837
Total	512,013	53,203	

Table 5. Amok (Test 1) - Anomaly Detection Metrics

False Positives	42,928
True Positives	10,275
True Negatives	511,451
False Negatives	562

Table 6. Amok (Test 1) - Anomaly Detection Metrics (Contd.)

Accuracy	92.3056%
Error Rate	7.6944%
Sensitivity	19.3128%
Specificity	99.8902%
Precision	19.3128%

Recall	94.8140%

Amok (Test 2) KNN Parameters- smote.k = 1, knn.k = 1, smote.perc.over = 120, smote.perc.under = 200. Amok (Test 2) - Confusion Matrix- N = 565,216

Table 7. Amok (Test 2)

	Predicted: NO	Predicted: YES	Total
Actual: NO	529,906	24,473	554,379
Actual: YES	1099	9,738	10,837
Total	531,005	34,211	

Table 8. Amok (Test 2) - Anomaly Detection Metrics

False Positives	24,473
True Positives	9,738
True Negatives	529,906
False Negatives	1099

Table 9. Amok (Test 2) - Anomaly Detection Metrics (Contd.)

Accuracy	95.4757%
Error Rate	4.5242%
Sensitivity	2.8464%
Specificity	99.7930%
Precision	28.4645%
Recall	89.8588%

Deauthentication (Test 1) KNN Parameters- Smote.k = 3, knn.k = 5, smote.perc.over = 150, smote.perc.under = 90. Deauthentication (Test 1) - Confusion Matrix- N = 558,167

Table 10. Deauthentication (Test 1)

	Predicted: NO	Predicted: YES	Total
Actual: NO	512,542	42,022	554,564
Actual: YES	95	3,508	3,603
Total	512,637	45,530	

Table 11. Deauthentication (Test 1) - Anomaly Detection Metrics

False Positives	42,022
True Positives	3,508
True Negatives	512,542
False Negatives	95

Table 12. Deauthentication (Test 1) - Anomaly Detection Metrics (Contd.)

Accuracy	92.4544%
Error Rate	7.5455%
Sensitivity	7.7048%

Specificity	99.9814%
Precision	7.7048%
Recall	97.3633%

Deauthentication (Test 2) KNN Parameters- smote.k = 1, knn.k = 1, smote.perc.over = 90, smote.perc.under = 400. Deauthentication (Test 2) - Confusion Matrix- N = 558,167

Table 13. Deauthentication (Test 2)

	Predicted: NO	Predicted: YES	Total
Actual: NO	527,780	26,784	554,564
Actual: YES	379	3,224	3,603
Total	528,159	30,008	

Table 14. Deauthentication (Test 2) - Anomaly Detection Metrics

False Positives	26,784
True Positives	3,224
True Negatives	527,780
False Negatives	379

Table 15. Deauthentication (Test 2) - Anomaly Detection Metrics (Contd.)

Accuracy	95.1335%
Error Rate	4.8664%
Sensitivity	10.7438%
Specificity	99.9282%
Precision	10.7438%
Recall	89.4809%

Authentication Request (Test 1) KNN Parameters- Smote.k = 3, knn.k = 5, smote.perc.over = 150, smote.perc.under = 90. Authentication Request (Test 1) - Anomaly Detection Metrics- N = 555,805

Table 16. Authentication Request (Test 1)

	Predicted: NO	Predicted: YES	Total
Actual: NO	513,668	40,945	554,613
Actual: YES	31	1,161	1,192
Total	513,699	42,106	

Table 17. Authentication Request (Test 1) - Anomaly Detection Metrics

False Positives	40,945
True Positives	1,161
True Negatives	513,668
False Negatives	31

Table 18. Authentication Request (Test 1) - Anomaly Detection Metrics (Contd.)

Accuracy	92.6276%
Error Rate	7.3723%
Sensitivity	2.7573%
Specificity	99.9939%
Precision	2.7573%
Recall	97.3993%

Authentication Request (Test 2) KNN Parameters- Smote.k = 1, knn.k = 1, smote.perc.over = 100, smote.perc.under = 300. Authentication Request (Test 2) - Anomaly Detection Metrics- N = 555,805

Table 19. Authentication Request (Test 2)

	Predicted: NO	Predicted: YES	Total
Actual: NO	540,840	13,773	554,613
Actual: YES	152	1,040	1,192
Total	540,992	14,813	

Table 20. Authentication Request (Test 2) - Anomaly Detection Metrics

False Positives	13,773
True Positives	1,040
True Negatives	540,840
False Negatives	152

Table 21. Authentication Request (Test 2) - Anomaly Detection Metrics (Contd.)

Accuracy	97.4946%
Error Rate	2.5053%
Sensitivity	7.0208%
Specificity	99.9719%
Precision	7.0208%
Recall	87.2483%

5 Conclusion and Future Work

This paper used the k-nearest-neighbor classifier to predict if data from the AWID dataset was normal or an attack. Our results were mixed with either high accuracy and high recall, high accuracy, and lower recall, or high recall and lower accuracy. Most of our time was spent on deciding how to pre-process the data, deciding which features to use with the classifier, and deciding which classifier and oversampling method to use. We learned a lot from the paper but if we could do it again we would select more features to classify with. In a future version, we would like to try using a classifier with the 3 variables we used plus the wlan_ra (MAC address) variable that we didn't use to see how it affects accuracy and recall for the 3 attacks that had lackluster results. To do this we would have to use plain oversampling instead of SMOTE, one-hot-encoding,

and probably use a different classifier such as neural network which is compatible with one-hot encoding. Additionally, we would like to test a more generalized model that simply determines if an attack occurred. By generalizing the model, the performance would most likely be improved.

References

1. Ranga, M. a, Venkatesh, M. L. H., & Venkanna, M. (2012). Design and Implementation Of AMBA-AXI Protocol Using VHDL For Soc Integration. International Journal of Engineering Research and Applications (IJERA), 2(August), 1102–1106.
2. Bisht, Prithvi. "Notamper: automatic BlackBox detection of parameter tampering opportunities in web applications." Proceedings of the 17th ACM conference on Computer and communications security. 2010.
3. Soleimani, Hamed, Mohmmad Ali Hadavi, and Arash Bagherdaei. "WAVE: Black-Box Detection of XSS, CSRF and Information Leakage Vulnerabilities." 2017 14th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC). IEEE, 2017.
4. Takamatsu, Yusuke, Yuji Kosuga, and Kenji Kono. "Automated detection of session management vulnerabilities in web applications." 2012 Tenth Annual International Conference on Privacy, Security, and Trust. IEEE, 2012.
5. Kumar, Rahul, and Aakash Kumar Goel. "Automated Session Fixation Vulnerability Detection in Web Applications using the Set-Cookie HTTP response header in cookies." Proceedings of the 7th International Conference on Security of Information and Networks. 2014.
6. Nadar, Virginia Mary, Madhumita Chatterjee, and Leena Jacob. "A Defensive Approach for CSRF and Broken Authentication and Session Management Attack." Ambient Communications and Computer Systems. Springer, Singapore, 2018. 577-588.
7. Dahse, Johannes, and Thorsten Holz. "Static detection of second-order vulnerabilities in web applications." 23rd {USENIX} Security Symposium ({USENIX} Security 14). 2014.
8. Aliero, Muhammad Saidu, et al. "Detection of structure query language injection vulnerability in a with target web-driven database application." Concurrency and Computation: Practice and Experience (2020): e5936.
9. Ze, Wang. "Design and Implementation of Core Modules of WEB Application Vulnerability Detection Model." 2019 11th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA). IEEE, 2019.
10. Jovanovic, Nenad, Christopher Kruegel, and Engin Kirda. "Pixy: A static analysis tool for detecting web application vulnerabilities." 2006 IEEE Symposium on Security and Privacy (S&P'06). IEEE, 2006.
11. Marashdih, Abdalla Wasef, and Zarul Fitri Zaaba. "Detection and removing cross-site scripting vulnerability in PHP web application." 2017 International Conference on Promising Electronic Technologies (ICPET). IEEE, 2017.
12. Medeiros, Ibéria, and Nuno Neves. "Effect of Coding Styles in Detection of Web Application Vulnerabilities." 2020 16th European Dependable Computing Conference (EDCC). IEEE, 2020.
13. Lukanta, Raymond, Yudistira Asnar, and A. Imam Kistijantoro. "A vulnerability scanning tool for session management vulnerabilities." 2014 International Conference on Data and Software Engineering (ICODSE). IEEE, 2014.

14. Yuan, Hui, et al. "Research and Implementation of Security Vulnerability Detection in Application System of WEB Static Source Code Analysis Based on JAVA." The International Conference on Cyber Security Intelligence and Analytics. Springer, Cham, 2019.
15. Anbiya, Dhika Rizki, Ayu Purwarianti, and Yudistira Asnar. "Vulnerability Detection in PHP Web Application Using Lexical Analysis Approach with Machine Learning." 2018 5th International Conference on Data and Software Engineering (ICoDSE). IEEE, 2018.
16. Jeevitha, R., and N. Sudha Bhuvaneswari. "Malicious node detection in VANET Session Hijacking Attack." 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT). IEEE, 2019.
17. Pan, Jinkun, and Xiaoguang Mao. "Detecting dom-sourced cross-site scripting in browser extensions." 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017.
18. Mokbal, Fawaz Mahiuob Mohammed, et al. "MLPXSS: an integrated XSS-based attack detection scheme in web applications using multilayer perceptron technique." IEEE Access 7 (2019): 100567-100580.
19. Kao, Da-Yu, Chung-Jui Lai, and Ching-Wei Su. "A Framework for SQL Injection Investigations: Detection, Investigation, and Forensics." 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2018.
20. Moustafa, Nour, Gaurav Misra, and Jill Slay. "Generalized outlier gaussian mixture technique based on automated association features for simulating and detecting web application attacks." IEEE Transactions on Sustainable Computing (2018).
21. Alswailem, Amani, et al. "Detecting phishing websites using machine learning." 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS). IEEE, 2019.
22. Wang, Qian, et al. "Access Control Vulnerabilities Detection for Web Application Components." 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC), and IEEE Intl Conference on Intelligent Data and Security (IDS). IEEE, 2020.
23. Deepa, G., et al. "Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications." International Journal of Information Security 17.1 (2018): 105-120.
24. Kurniawan, Aditya, et al. "Static Taint Analysis Traversal with Object-Oriented Component for Web File Injection Vulnerability Pattern Detection." Procedia Computer Science 135 (2018): 596-605.
25. Figueiredo, Alexandra, Tatjana Lide, and Miguel Correia. "Multi-Language Web Vulnerability Detection." 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, 2020.
26. Ibarra-Fiallos, Santiago, et al. "Effective Filter for Common Injection Attacks in Online Web Applications." IEEE Access 9 (2021): 10378-10391.
27. Choi, Hyunsang, et al. "HXD: Hybrid XSS detection by using a headless browser." 2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT). IEEE, 2017.
28. Chatzoglou, Efstratios, Georgios Kambourakis, and Constantinos Kolias. "Empirical Evaluation of Attacks Against IEEE 802.11 Enterprise Networks: The AWID3 Dataset." IEEE Access 9 (2021): 34188-34205.
29. Address Resolution Protocol From - Wikipedia, the free encyclopedia, accessed 27 June 2021, from https://en.wikipedia.org/wiki/Address_Resolution_Protocol
30. University of the Aegean - AWID3 datasets including Krack and Kr00k. It is available in pcap format, accessed 1 July 2021, from https://icsdweb.aegean.gr/awid/download-dataset.