Automatically Constructing Custom Network Security Datasets with Word Embeddings

Alexander Korobchuk, *University of Colorado Colorado Springs, akorobch@uccs.edu,*

Abstract

A common challenge in any machine learning task is constructing the proper dataset. This challenge becomes greater when the task is for network security, such as for an intrusion detection system. For instance, hand picking features from network packets to use in classification tasks can be a strenuous process. Furthermore, every network is different, therefore it is difficult to create a dataset that can represent most networks. Thus, there is a need for the ability to easily create custom datasets that are tailored to a specific network. In this research, the potential of creating datasets by automatically extracting features from network packets is explored. Attacks are simulated by using common enumeration tools utilized by hackers, while also capturing all the packets on the network. The packet data is preprocessed and used to train a Word2Vec neural network model, of which features are automatically extracted from each packet and compiled as a vector representation. Using the vectors, a dataset can be created, thus being a simple means to formulate a custom-tailored dataset to a network. The results are shown by applying the datasets to a logistic regression machine learning model for the classification task of malicious and benign network packets.

Index Terms

Word2Vec, machine learning, neural networks, network packets

I. Introduction

For any machine learning task, creating the proper dataset is arguably the most important part. Features must be selected that will allow the model to accomplish the desired task. For instance, the example of predicting housing prices can be used. The features for such task could be square footage, number of bedrooms, number of bathrooms, and of course sale price. But then over and underfitting must be taken into consideration. If there are too many features, there could be an overfit. If there are too little, there could be an underfit. As a result, this could be months of work collecting features to a loss.

For less trivial tasks, it may be a lot more difficult to hand pick features. An example of this could be for network security. Deciding which features to select out of something like network packets could be a very strenuous process. Furthermore, every network may look different. How could one select features from network packets and be confident those features apply to all networks?

That is where this research comes in. Here, the idea of automatically creating network datasets that are custom tailored to a network via neural networks is explored. The basic outline is as follows: simulate non-damaging network attacks on a network while capturing these packets, use the packets to train the Word2Vec neural network model, extract the weights from the model and construct the dataset. Then this dataset may be used for classification tasks such as predicting if a packet is malicious or not.

II. Packet Capture

The first step is to simulate attacks while capturing all the packets on the network. A program was written that automatically performs attacks as well as captures packets. For all the trails in the research, the process was the same: start a packet capture and wait thirty seconds. After the time has passed, the program begins to utilize the attack for the specific trial. The program runs for five minutes and then discontinues the capture.

Once this process has completed, the program creates a pcap file (packet capture) of which contains all the packets, both malicious and benign. The program also creates a file that contains each packet number that was malicious. This is determined by looking at the IP address of the sender in each packet. If the IP address is the same of the computer running the program, then it is determined that the packet is malicious. This file is extremely important for our dataset because this is how we will determine whether to mark a packet as malicious or benign. It is also important to note that we are assuming that the remaining packets are benign.

III. Packet Processing

Next, there are some important things that must be done before the Word2Vec neural network model can be trained. Each packet in the pcap file contains a large array of information. This information must be processed so that the neural network model can understand what is being inputted. The process is simple: first remove anything in the packet that isn't a letter or a number. Numbers are kept because they can describe the size of the packet or the destination port. Then the entire packet is lowercased.

Finally, the source and destination IP address is removed. This is important because these are items that a machine learning model could pick up on. The source IP address will always be the attackers IP and if the machine learning model can determine this then that would result in 100 percent accuracy. This is the same for the destination address.

This processing is done on every single packet that was captured in the pcap file. Once it has been completed, this information can be fed into the Word2Vec neural network model.

IV. Training the Word2Vec Model

Word2Vec is a neural network model that has the ability to create word embeddings[1]. A word embedding is a vector representation of a word. This is also called a word vector. Word vectors can be plotted in space and a common use of these vectors is for word prediction and similarity. For example, Word2Vec can be trained with many articles and documents found on the internet, and then similarity of words can be calculated. But it's important to note that these similarities are purely based on the articles given to the model.

The classic example uses the three words: king, man, and woman[2]. Using these words and their word vectors, mathematical operations can be performed. If the vector of king was subtracted by the vector of man, and then the vector of woman was added, the resulting vector would be roughly equal to the vector of queen. This is an example of how these word vectors can be manipulated.

$$king - man + woman = queen$$

Although for this application, this is not useful. Knowing the similarity between the words "packet" and "destination" is of no use for our datasets. But these word vectors can still be used, as long as a little bit of more work is done. The issue is that Word2Vec returns a vector representation of each word *individually*. More processing must be done in order to take these individual word vectors and create an overall representation of each packet.

This can be done by averaging all of the vectors in the packet. For example, let's say that a packet has 350 words and each vector is 5 dimensional. We go through each word starting at its first vector index, and then perform a summation of every words first index. Finally, we divide that total by total number of

words in that packet (in this example, 350). This gives us the first index of our new average vector. We perform this operation four more times and the result is an average vector that can be used to represent the entirety of a packet. Using this process, the datasets can finally be created.

V. Creating the Dataset

To start, we will look into the dimensions of our dataset. The number of rows in the vector representation of a packet will define how many features the dataset will have. For example, if the vector of each packet contains 100 rows, then the dataset will contain 100 features. Similarly, if there are 10,000 packets in a pcap file, there will be 10,000 entries in the dataset.

Since the program creates column vectors, I then added a feature to convert these into row vectors. After being converted, the program checks the packet number with the packet identification file that was created earlier. If the packet was marked as malicious, the program will append a 1 at the end of the vector. If the packet was not marked, then it will append a 0. After doing this for every single packet, it has finally created the dataset.

VI. Testing

For testing this process, multiple trials were ran. There were two machines, one running an Apache webserver and another that would perform the scans and do the packet capture. This second machine is what is performing attacks and then creating the dataset. There were five trials using the following enumeration tools: Nmap[3], Nikto[4], and Dirb[5]. These tools scan webservers for vulnerabilities and information such as directories the user should not be able to see as well as various exploits for the server depending on its version. Figure one shows the number of packets captured with each trial, including the malicious ones.

It's important to note that the 'nmap stealth' trial did have malicious packets, but there were so few that the graph simply cannot display it. The average percent of malicious packets throughout all the trails was 9.7. The aim was to keep this number as low as possible because there are millions of packets running through a network and if an attack were to happen, a very tiny percentage of the packets would be malicious. So when training the model, it was desired that the model see few malicious packets but hope that it can still classify them with a high precision.

When it came to creating the datasets, it was important for me to figure out what the minimum amount of features should be. So, for each trial, I created datasets with 10, 25, 50, and 100 features. Once these datasets were created, they were used to train a logistic regression machine learning model. This was the method of determining how many features are necessary as well as the various metrics for determining if the dataset has trained the model appropriately.

VII. Results

For concluding results, it's important to note that accuracy was not used as a metric. This is due to such a small amount of packets being malicious. Our model could simply guess that every single packet was benign and thus achieve a high accuracy and this of course is not desirable. So the first metric we will look at is precision. Precision is defined as:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Here, higher is better. This determines how well our model can classify malicious packets. This number is more valuable than accuracy in our case because we want to know if our model can pinpoint the malicious packets, even if there are so few of them in the dataset. Figure three displays the precision obtained with the trials.

The highest precision was obtained by the dataset that included fifty features for each packet (as seen in figure three). The precision began somewhat low at ten features, and then increased greatly to twenty

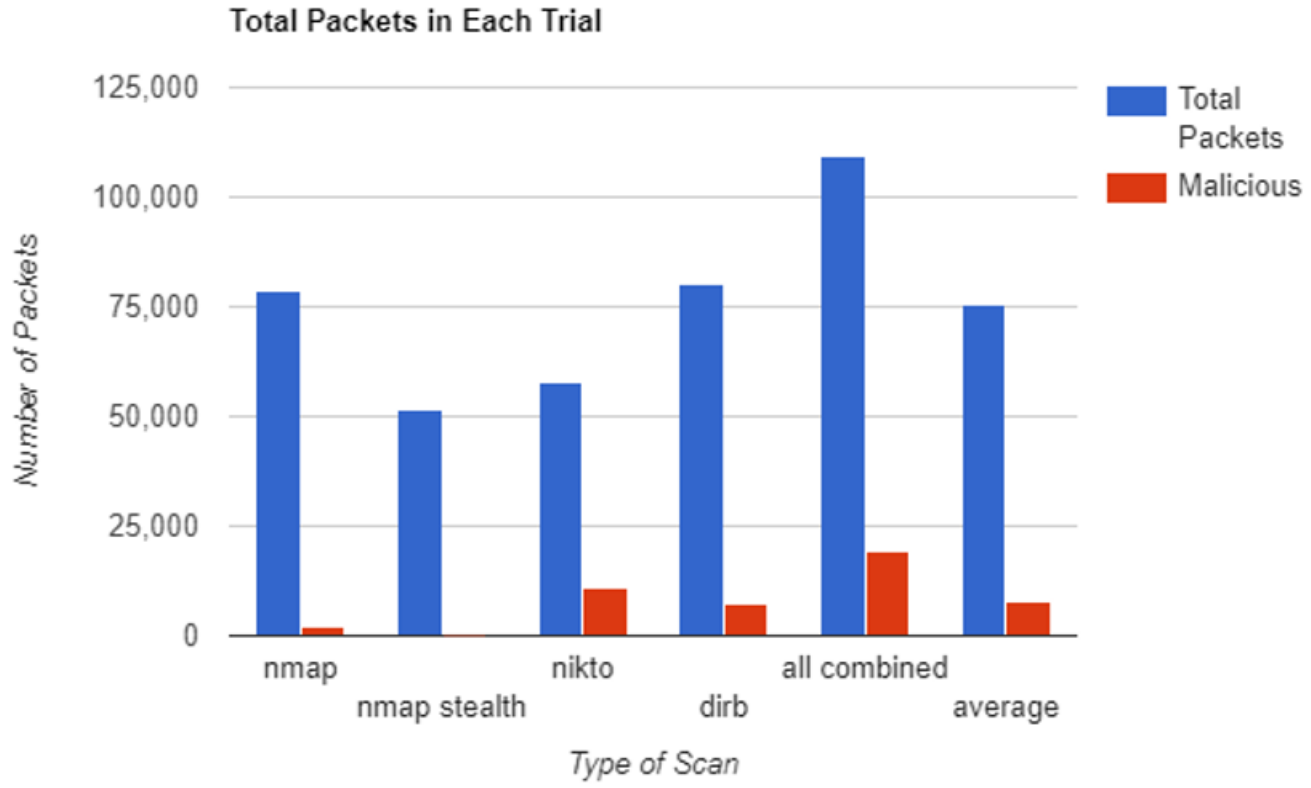

Fig. 1. Total number of packets captured in all the trails including the average

five and to fifty as well. But actually dropped slightly at one hundred features. This indicates that fifty features is adequate enough for a machine learning model to learn from and to be able to precisely classify malicious packets.

The next metric that we want to take a look at is recall. Recall is another important measurement to observer in this particular situation once again because of the so few malicious packets in a dataset. Recall is defined as the following:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Recall is the ratio of relevant items that are successfully obtained. In other words, the higher the recall, the better the model is at actually obtaining malicious packets to learn off of. A lower recall would mean that the model did not have many malicious packets to test on. This is something that needs to be avoided because of the so few malicious packets in each dataset. So of course a higher recall is desired. Figure four shows the recall of each trial as well as the vector size of each packet and figure five shows the average ratio achieved for the recall on all trails.

For recall, the highest ratio was obtained by the dataset with 100 features. It's important to note here that the recall increases very slightly from 50 to 100 features. But, similarly to precision, the recall increases greatly from 10, to 25, and then to 50. This results in the conclusion that even though 100 features beat 50, 50 is still the better number. The reason for this is because the metric begins to plateau after 50 features. Although it slightly increases, it's clear that a plateau is in sight. The performance costs for making a 100 feature dataset outweighs the few percentage points increase we get when we increase to 100 features. Also, the increase is so slight considering we are doubling the amount of features we have.

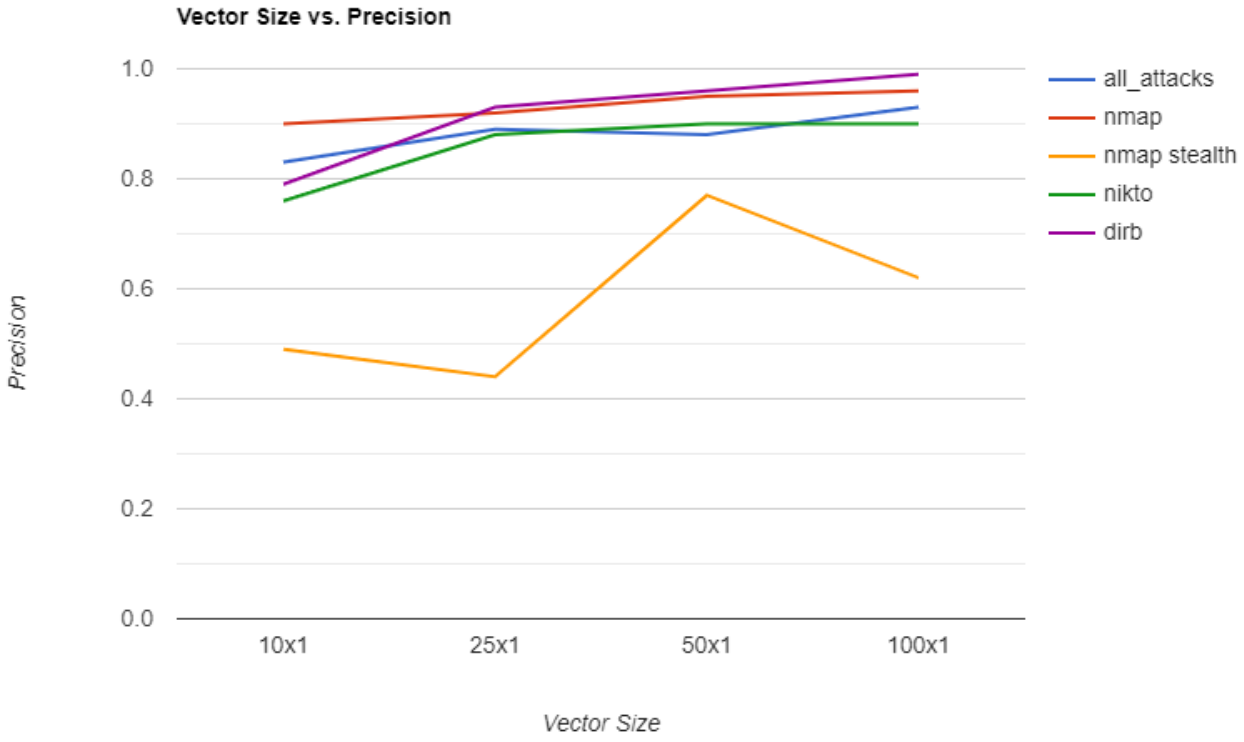

Fig. 2. Precision of all trials, including the vector size of each packet

10x1 Precision (average)	25x1 Precision (average)
0.754	0.812

50x1 Precision (average)	100x1 Precision (average)
0.892	0.880

Fig. 3. Precision of the various feature sizes

So therefore it does not make sense for us to use 100 features. This is why we conclude that 50 features is once again adequate.

Finally, we will take a look at the F Score. This is simply another measure of a tests accuracy. It can

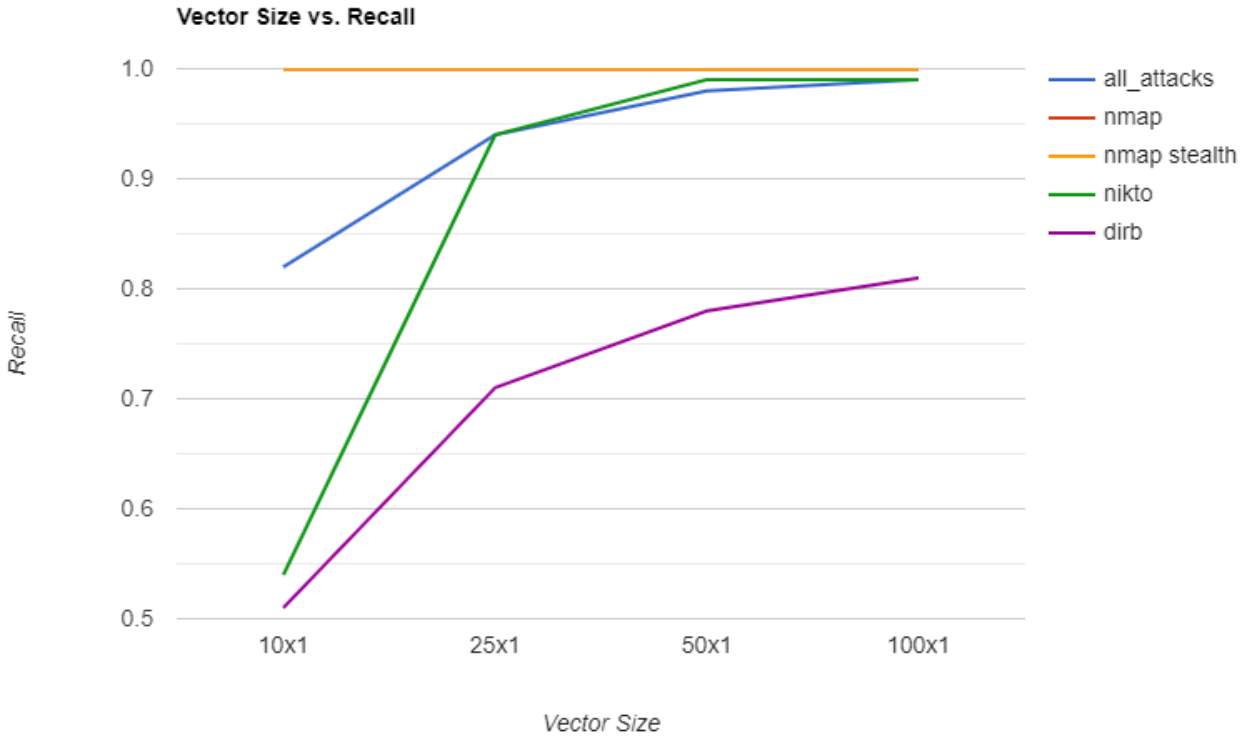

Fig. 4. Recall of all trials, including the vector size of each packet

10x1 Recall (average)	25x1 Recall (average)
0.744	0.918

50x1 Recall (average)	100x1 Recall (average)
0.95	0.958

Fig. 5. Recall of the various feature sizes

be a useful metric to look at because it takes in account of both the accuracy as well as the recall. Since the F Score takes in account of both the recall and the precision, it can easily be gathered that this metric somewhat represents both the precision as well as the recall of our model. The F Score uses the following

formula:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

The F Score is often used to measuring the performance pertaining to searching and classification. Some criticize the use of the F Score because it gives equal importance to both the recall as well as the precision. For example, the model may have extremely high precision but low recall and thus the F Score would not be very great. Precision is certainly more important than recall but the F score would give the impression that the model was not trained adequately. So it's important to take the F Score with a grain of salt.

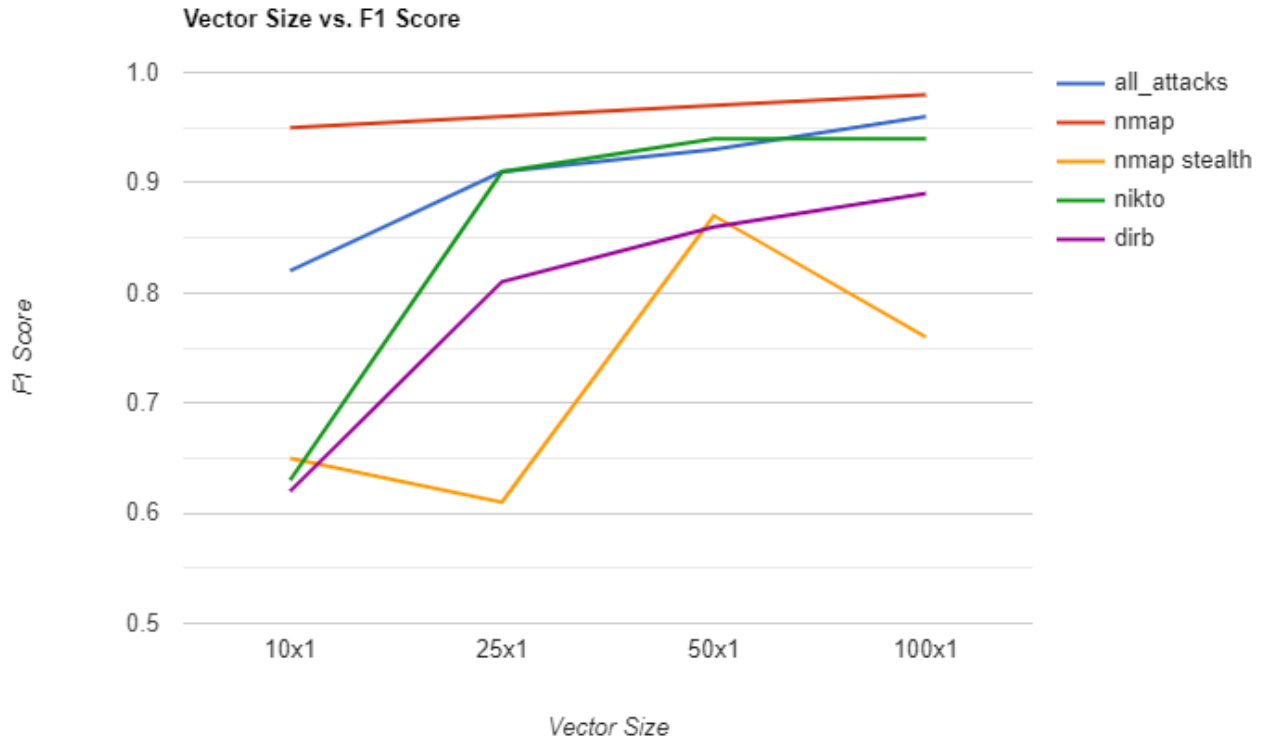

Fig. 6. F1 Score of all trials, including the vector size of each packet

Figure six shows the F Score obtained from all the trails as well as their vector sizes. Figure seven shows the average ratio of the F Score. Once again, the fifty feature datasets stand out from the rest, obtaining the highest F Score on average. This indicates that fifty features is adequate. Similarly to the previous metrics, the F Score begins to plateau at 50 features, and actually begins to decline at 100 features. This is once again important to note because the performance impact 100 features has on a computer is vast. We are doubling from 50 to 100 features and as a result the time that it takes to create the datasets increases greatly. It's also important to mention that F1 Score should be observed on a case by case scenario. In our specific situation, we had similar precision ratios as well as recall ratios therefore it is appropriate to observe the F1 Score. It's also appropriate to desire higher F1 Scores. So this is why this metric is observed here and why we declare that the 50 feature dataset as the best option in terms of recall.

10x1 F Score (average)	25x1 F Score (average)
0.734	0.84

50x1 F Score (average)	100x1 F Score (average)
0.914	0.906

Fig. 7. Recall of the various feature sizes

VIII. Conclusion

Machine learning and deep learning is something that isn't deployed much in the cybersecurity domain. I believe this is due to the complexity, and more importantly, the uniqueness in every network. That is why I wanted to explore the option of creating custom tailored network security datasets. Although it may seem quite far fetched, it could be feasible to attack a network in a controlled manner, while also capturing all of the packets to then create a dataset. This dataset would know what the normal traffic of the network looks like but also what a malicious packet may look like.

Based on the results presented here, two conclusions can be made. First of all, it could be a possibility to create custom tailored network datasets using these methods. A more advanced program could be written that could commit a variety of attacks and capture packets as well as the attacks. Using these, the datasets can be created.

Also, an appropriate vector size of each of the entries in the dataset should be no less than 50 features. It has become apparent through this research that 50 features is indeed enough information for a machine learning model to learn from. If more features are added, there is a severe tradeoff with performance and time it takes to create these datasets. Furthermore, the metrics barely increased. So there is not a good trade in performance for results because the results are so suttle after 50 features.

IX. Future Work

There are a few possibilities with this work. Firstly, it would be interested to explore the idea of placing some modified version of this system behind a traditional intrusion detection system (IDS) as another line of defense. IDS' are known to produce a large array of false positives[6], but they can work somewhat fast. If the system presented in this paper were to be placed behind an IDS, it could be a possibility to feed the false positives produced by the IDS to this system and allow for the system to check those packets individually. Of course the system would first need to be trained, but then it could be use to grab the packet data of the false positives (or any positives made by the IDS) and double check for the validity of the classification. That way, if the IDS casts a positive classification on a packet, before alerting the IT team, the system presented in this paper could then check that positive as a second line of defense. This could greatly reduce false positives.

X. Related Work

This work was greatly inspired by an approach called Packet2Vec[7]. In their work, they also use a Word2Vec approach in order to automatically extract features from a packet capture. The differences are that they focused on a pre-made dataset (the 2009 DARPA network dataset), whereas here we focus on creating our own datasets that are custom tailored to a network. They also chose to remove the port information of each packet. In my research, I chose to keep the port information because I believe that this could give clue to whether a packet is malicious or not.

They also chose to focus a bit more on optimization. In my research, I did not touch optimization as it was out of scope.

References

[1] "Word2vec," Wikipedia, 24-Oct-2020. [Online]. Available: https://en.wikipedia.org/wiki/Word2vec. [Accessed: 04-Dec-2020].
[2] P. Migdał, "king - man + woman is queen; but why?," Piotr Migdał - blog, 06-Jan-2017. [Online]. Available: https://p.migdal.pl/2017/01/06/king-man-woman-queen-why.html. [Accessed: 04-Dec-2020].
[3] Nmap. [Online]. Available: https://nmap.org/. [Accessed: 04-Dec-2020].
[4] "CIRT.net," Nikto2 — CIRT.net. [Online]. Available: https://cirt.net/Nikto2. [Accessed: 04-Dec-2020].
[5] "DIRB," Penetration Testing Tools. [Online]. Available: https://tools.kali.org/web-applications/dirb. [Accessed: 04-Dec-2020].
[6] Chuck Brooks Dec 3, "False Positives in IDS: Irritating But Often Necessary," ATamp;T Cybersecurity. [Online]. Available: https://cybersecurity.att.com/blogs/security-essentials/false-positives-in-ids-irritating-but-often-necessary. [Accessed: 04-Dec-2020].
[7] E. L. Goodman, C. Zimmerman, and C. Hudson, "Packet2Vec: Utlizing Word2Vec for Feature Extraction in Packet Data.," 2019.