Ruby Link and Anoushka Sarma
Nicholas Lahaye
CPSC-393-01
16 December 2023

Final Report: Detecting Malware

Problem

Cybersecurity is an issue that gains popularity as technology advances and hackers become more creative. Malware and malware detection is a growing field. Malware is a type of harmful software with the intention of destroying, disrupting or harming computer systems. A lot of times it is used to gain unauthorized access to data and other criminal activities. There are many types of malware such as viruses, worms, adware, and scareware. In current news, there have been many malware attacks that have done significant damage such as the China Google Attack of 2009 where Chinese hackers attempted to hack into google accounts of human activists. Although they were not successful, this was shocking as emails contain a lot of private and secure information. Malware detection is important because it helps to improve network connectivity and security and create future preventative measures.

Currently there are many solutions to malware detection such as Antivirus Software, Endpoint Protection Platforms, Sandboxing, and Cloud Based Security Systems. Antivirus softwares uses a database of known malware signatures and scans incoming files to compare and filter out malware. Endpoint Protection Platforms combine antivirus softwares, firewalls, and other detection softwares to filter and screen for malware packets. Sandboxing looks at malware in a controlled environment to learn about the behavior. Cloud based security services utilize the cloud infrastructure for scalability and to detect real time threats and provide solutions.

The solution that we are proposing is using a machine learning feed forward neural network to assess packet attributes and learn what specific specifications lend itself to malware. Our model would be able to grow and evolve as neural networks can learn and retrain as new data is added.

Data

The dataset that we are using was taken from Kaggle but is originally from a GitHub repository. The data was scraped by CICFlowMeter which is a tool that generates Biflows from Packet Capture files, and extracts information from these flows. The data is used to detect congestion, malware and other abnormalities in the network. The original dimensions of the dataset are 355,626 observations and 86 different features. After feature selection, the same number of observations are retained but 24 features were chosen. The input data consists of 7 continuous variables and 16 categorical features. The classes of the data are Adware, SMS Malware, Scareware, Benign. Adware is malware that displays unwanted pop-up ads. Scareware is malware that tricks the user into downloading software that they don’t need. SMS attacks

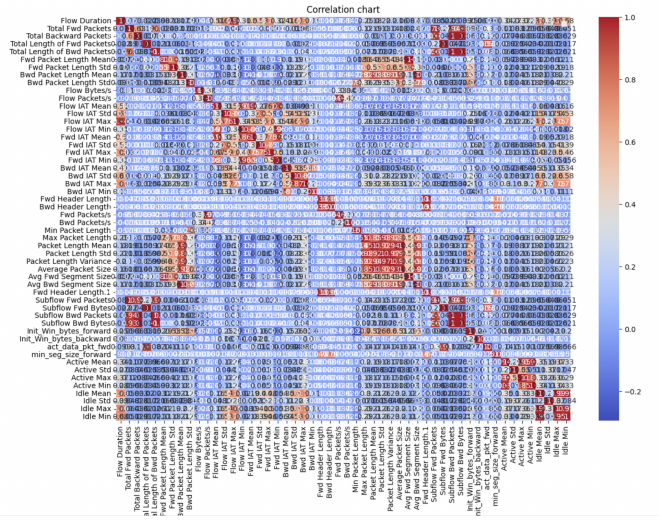
involve the creation and distribution of malware by cybercriminals designed to target a victim's mobile device. The features used in this model are as follows:

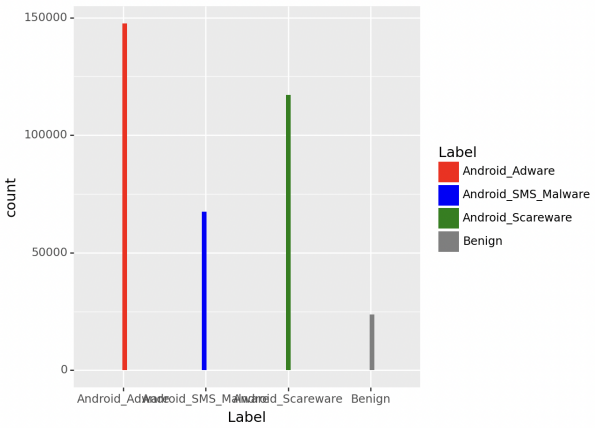
Source IP - IP address
Protocol - rules of transmission
Packet Length Mean - in bytes
Flow Duration – Duration in Microseconds
Total Fwd Packets - Total packets out
Total Length of Fwd Packets - size
Fwd IAT Mean - Mean time between two packets in forward direction
Bwd IAT Mean - Mean time between two packets in backward direction
Flow Bytes/s -bytes per second
Flow Packets/s - packets per second
Active Mean - mean time before becoming idle
Fwd Header Length - bytes in header
FIN Flag Count - close a connection
SYN Flag Count - synchronizes sequence numbers
RST Flag Count - doesn't comply with protocol criteria
PSH Flag Count - deliver data quickly
ACK Flag Count - acknowledgement
URG Flag Count - prioritize data
ECE Flag Count - congestion indication
Init_Win_bytes_forward - initial window forward
Init_Win_bytes_backward - initial window backward
Act_data_pkt_fwd - packets with at least 1 byte of TCP data payload
Idle Mean - idle before becoming active
Source Port - port being used to send packet

Exploratory Analysis and Preprocessing

When exploring the data we found that many of the continuous input variables are highly skewed and categorical variables have uneven class distributions. We plotted histograms of many of the variables to view the distribution of values. The trend of imbalance includes the dataset label or outcome variable. Adware has the most observations with 147,443, then Scareware with 117,081, Malware at 67,394, and Benign, or no malware, with 23,708. It is important to note that the overall trends of imbalance in most of the variables could lead to skewed/suboptimal results. Additionally, when looking at the correlation between variables, it was found that many of the variables were highly correlated such as the forward and backward time variables. We removed many of the variables that were highly correlated, but we did take into account that because this dataset looks at both forward and backward flows to detect malicious activity, high correlation doesn’t mean that they will give us the same information. Slight variation in the forward to

backward direction could have a big influence on whether or not malware is detected. To prepare the data for our models, we preprocessed the categorical variables. Many of the categorical variables have thousands of potential outcomes. We still wanted to include these so we arranged them into buckets with the most common values (usually claiming over 80% of the distribution) having their own buckets and the other less common ones grouped into one. We did decide, however, to treat some of the categorical variables as continuous values to see if the model could still pick up on some trends. We used MinMaxScalar to make sure that everything going into the model could be compared on the same scale.





The figures above show a correlation matrix between all the variables and the distribution of the labels. The ones colored in red are positively correlated and darker blues are strongly negatively correlated.

Models

3 models were implemented. The first model was a simple feed forward neural network with a single dense layer and 256 nodes. Ten epochs were used with a batch size of 1024. The Adam optimizer and categorical cross entropy loss function were used during training. We also tested this model with rmsprop for the optimizer, but Adam performed better so we decided to stick with it for the remainder of the models. To measure the performance of the model, the accuracy and loss metric was calculated. We wanted to see how well the model would perform given the simplest neural network model and then we could add more complexity as we went on.

The second model builds on the first implementation but adds two additional dense layers to add more complexity to the model. A smaller batch size of 512 was also implemented instead of the previous 1024 to see if the model converged quicker and also with more stability. The main goal

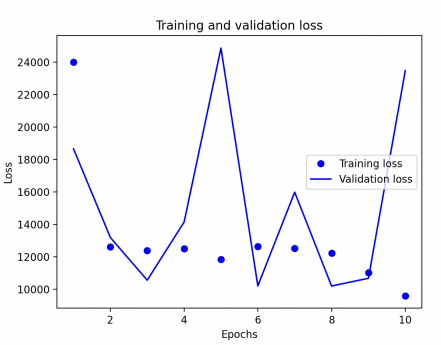
of this model is to see if a more complex model could pick up on the patterns better than a simple model.

The third model builds on the second implementation. We added in one more dense layer and also added L2 regularization. Regularization is used to ensure that the model isn't being overfit so that it can be better generalized to training and test data. Due to the fact that our model has many features, we decided to use L2 over L1 because L1 drags some weights to zero. With our knowledge of malware and traffic flow error detection, even the smallest changes in an exchange can indicate malicious activity. With this, we didn't want the regularization technique to completely get rid of weights for any features, however small.

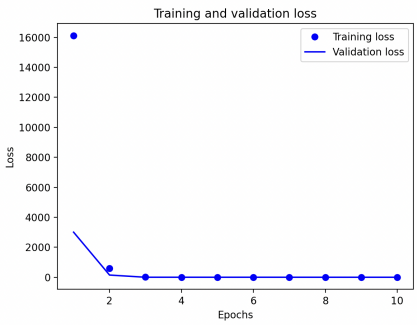
While not shown in our code, the first time that we implemented these three models we used a smaller subset of features and preprocessed them slightly differently than what our model currently shows. As discussed in our results section, we added in additional features such as protocol, Flow IAT min, and Fwd IAT min and decided to treat Source Port as a continuous feature. We retrained the models with these adjustments to output the results presented in our code. While we didn't use transfer learning, we did take inspiration from other people's code who used this dataset. Someone had generated a random forest model and was able to depict the most important features that they found. We decided to add these additional features to our models.

Results

With our initial data excluding features like protocol, Flow IAT min, Fwd IAT, and Source Port, the first model had a high loss of 10466.578 and 25.9% accuracy. This model performed very poorly and was not able to classify the data correctly. The first figure on the right shows the training and validation loss for the first model. The graph depicts how poorly the model is performing. Each epoch is very noisy and

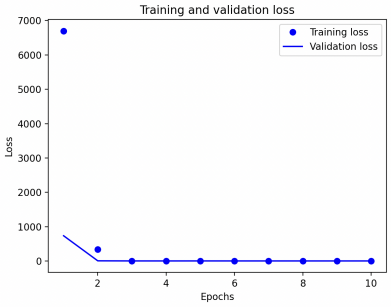


The second model had a loss of just 1.199 and 43.2% accuracy. The addition of a few dense layers significantly enhanced the performance of the model. While the accuracy is still fairly low this model performed better than the first one with training and validation loss being more constant across the training epochs. The second figure on the right shows the training and validation loss for the second model. This model performed much better in both metrics and



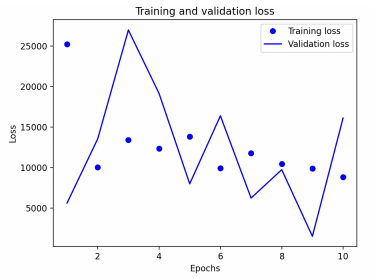
visually we can see that the convergence was much more stable for the loss.

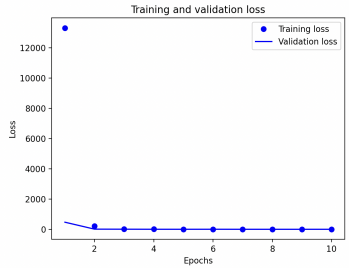
The third model had a loss of 1.655 and accuracy of 45.2%. While the loss increased by a small margin the accuracy improved by 2%, indicating that the addition of regularization was beneficial to the model. The figure to the right shows the training and validation loss for the third model. This graph indicates that the third model performed similarly to the second model in terms of reducing loss.

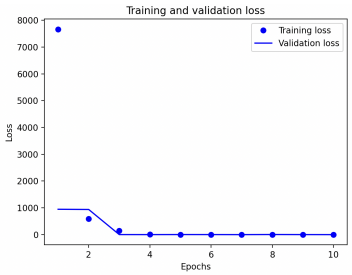


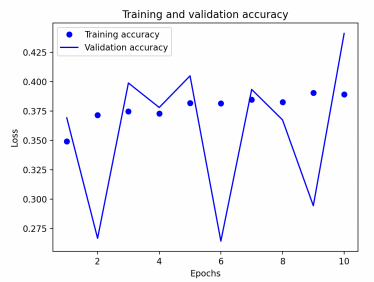
After training these initial models, we decided to go back and add more features that we thought may improve the performance. We had initially wanted to add variables, specifically the Source Port, but knew that it would be very time consuming and require a lot of computate to convert the variable into thousands of one hot encoded vectors. With this, we were advised to try implementing the feature as a continuous one to see if it improved the performance. With the addition of Protocol, Flow IAT min, Fwd IAT min and Source Port, we saw a big improvement in the performance of the models.

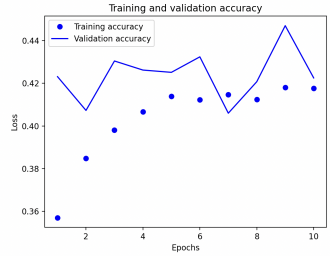
The first model was able to yield an accuracy of 40.8%, nearly 15 points higher than without these variables. However, the loss was still very high at 16107.207. The second model performed much better as well and the accuracy rose to 42.2% with a loss of 2.27. Finally, the third model performed the best with accuracy of 43.7% and a loss of only 1.773. The graphs below show the the validation and accuracy for all three models. The first model is on the left.

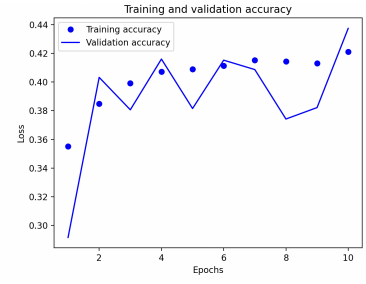










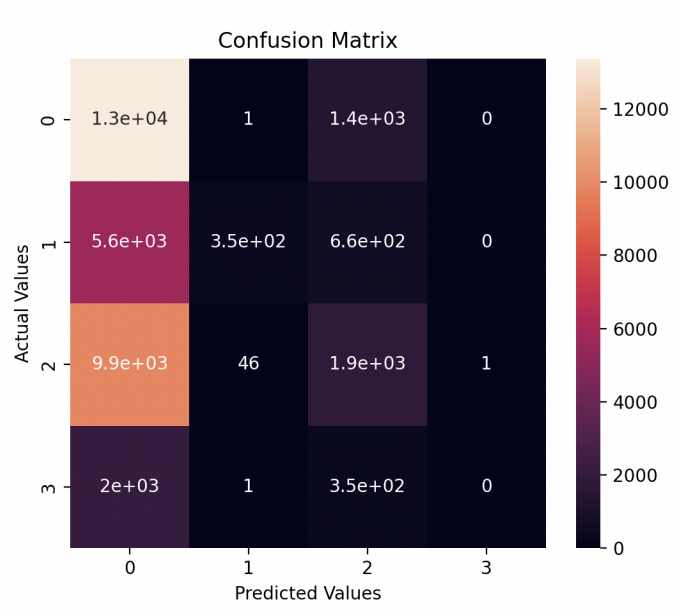


However, we were wondering why our loss was so low if our model was still performing poorly.

To investigate this, we also decided to calculate the precision and recall for each of the classes. When calculating the precision, recall and accuracy of the third model, the following results were calculated:

Precision: [0.43311524, 0.88059701, 0.43817079, 0]
Recall: [0.90528739, 0.05353092, 0.15932807, 0]

Precision is the calculation of how well the model did in calculating true positives out of all the positives predicted. For class 0 (Adware), it predicted a positive case 43% of the time, for class 1 (SMS Malware) it predicted positives correctly 88% of the time, class 2 (Scareware) was 44% and class 3 (Benign) 0%. The recall is how many positives the model was able to predict out of all the positive cases. For class 0 it was able to predict positives 90% of the time, 5.3% for class 1, 15% for class 2 and 0% for class 3. We can also see in the confusion matrix below that while Adware had the highest recall, it looks like it was also the most predicted class overall. Because Adware has the highest number of initial observations, the model is most likely to predict this value. For SMS Malware, visually we can see that it performed well. It has the highest precision, and visually we can see that the model is able to successfully classify SMS attacks while not falsely identifying it for one of the other classes. Scareware is similar to Adware in that this class is predicted a lot, likely due to the fact that this class has the second most observations in the dataset. The model does a very poor job at predicting class 3, Benign, which is likely causing the overall poor accuracy of the model while still maintaining a low loss. The model was unable to correctly assign class 3 at all and this could be due to the uneven distribution of the classes. It had the fewest number of observations by far and was only predicted once.



The figure above shows a confusion matrix of model 3. The diagonal represents the correctly predicted values. Class 3 had no correctly predicted values while class 0 had the highest volume of correct predictions. The highest volume of misclassifications were for classes 0 and 2. This heat map indicates that improvements can be made to the model to improve accuracy, precision and recall.

The results are meaningful because our model is able to semi-predict whether a packet is malware. The results are also telling us the weak spots in our model and where it is specifically performing poorly. One thing that may have contributed to the lower accuracy scores is the distribution of the class labels. With more data, all of the models would have more information on packet attributes that contribute to malware. Additionally, we didn't use all of the features and there may be some missing information that could increase the overall performance. For the information provided by the dataset, our models performed relatively well and we were able to significantly improve the performance over various tests.

Future Steps

While our model ultimately didn't perform well, immense improvements were made from our first model to our last. It would be ideal to have a very high accuracy for malware detection since malware can cause harmful and serious issues. We tried multiple optimizers and each model is an improvement on the last, however it may be that the data's distribution and information isn't enough to accurately predict what type of malware is present above our current accuracy. Perhaps the malware signatures and identification cannot be classified through the information only provided by the data. For future steps, there may be more complex models that can handle the data better and be more accurate in predicting the different classes of malware. Adding more data, specifically to the Benign class, may also help the model's accuracy as more information could be useful in detecting malicious software in packets.

Sources

Cyber Security Software and anti-malware. Malwarebytes. (2023, December 11). https://www.malwarebytes.com/

GitHub. (n.d.). https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt

Malware, Scareware, & ransomware. Information Security & Policy. (n.d.). https://informationsecurity.iu.edu/personal-preparedness/hardware-software/malware-scareware-ransomware.html

Kaspersky. (2023, April 19). *SMS attacks and mobile malware threats*. usa.kaspersky.com. https://usa.kaspersky.com/resource-center/threats/sms-attacks