Port Scanning Detection in Packet Capture (PCAP) File Using Decision Tree and Logistic Regression

Wang Mingye  
*WKWSCI*   
*Nanyang Technological University*Singapore  
mwang015@e.ntu.edu.sg

*Abstract*—In order to maintain the security of information systems, it is necessary for information system owners to detect malicious behaviors or cybercrime intrusions. However, some types of network malicious behaviors are hard to identify and monitor by humans or simple computer processes. Using a trained machine learning model will help us a lot since the machine learning model can judge whether a network behavior is malicious and can tell the type of this malicious behavior automatically with a network behavior record as the input. In this article, a decision tree model and a logistic regression model are separately trained on data extracted from a packet capture (PCAP) file, which contains the network behavior records of port scanning behavior and benign behaviors. The packet capture file is from CIC-IDS2017, an intrusion detection evaluation dataset. Both the trained models can tell whether a network behavior is a port scanning. The detailed processes of gathering the train data, preprocessing data and training models are included in this article. In theory, same methods can be applied to other PCAP files in CIC-IDS2017 that contain other types of malicious behavior, such as Brute Force, DDoS, XSS and Infiltration, but only Port Scanning dataset is used in this article due to the limitation of the author’s device hardware capability.

Keywords—Intrusion Detection, Machine Learning, Decision Tree, Logistic Regression, Port Scanning

# The forensic problem

Port scanning [1], a kind of malicious network behavior, is a method of determining which ports on a network are open and could be receiving or sending data. It is also a process for sending packets to specific ports on a host and analyzing responses to identify vulnerabilities. Port scanning can help to determine open or vulnerable server locations and diagnose security levels. Port scanning can also reveal the presence of security measures in place such as the firewall rule of the server.

This article aims to use machine learning models to find out the port scanning behaviors from a group of network behaviors whose information are extracted from a raw PCAP file. It is essentially a binary classification problem. That is, judging whether a network behavior is a port scanning or not.

# Survey of Related Work

## Methods to Conduct Intrusion Detection

As early as 1987, Denning, D. E. presented a model of a real-time intrusion-detection expert system capable of detecting break-ins, penetrations, and other forms of computer abuse is described. [2]

More recently, machine learning models are utilized to conduct intrusion detection. Almost all the machine learning algorithms that are able to do classification can be used for intrusion detection. Here are some examples: Liao, Y. & Vemuri, V. R. used KNN [3], Chen, W. H., Hsu, S. H., & Shen, H. P. used SVM [4] and Farnaaz, N., & Jabbar, M. A. used random forest [5].

Moreover, without doubt, deep learning models such as CNN and LSTM, which are also a kind of machine learning algorithms that are able to do classification, can be used to detect intrusion. Deep learning models can even directly take the binary content [6] of the entries in PCAP file as the input, rather than structured data, which release us from the feature engineering or data preprocessing.

Both the expert system and machine learning algorithms (deep learning is a subset of machine learning) are methods under artificial intelligence. Considering that intrusion detection in essence is a kind of prediction or judgement based on given features, which is exactly suitable to be solved via AI methods, no wonder most of published work used methods related to AI to conduct intrusion detection.

## Datasets of Intrusion Detection

Due to the volume of the internet nowadays, network intrusion behaviors can be ubiquitous. Therefore, there are many intrusion detection datasets whose data are collected from real life production network, though it is still possible to produce an artificial dataset via conducting specific attacks yourself to a network environment under your control.

Many intrusion datasets give structured or processed data rather than raw PCAP files. These data may be more similar to traditional machine learning or datamining datasets that are cell by cell, row by row and column by column. KDD Cup 1999 dataset [7] contains a wide variety of intrusions simulated in a military network environment. CTU-13 dataset [8] is a labeled dataset with botnet, normal and background traffic.

There are also datasets that are or includes PCAP files. ISCXIDS2012 [9] and CIC-IDS2017 [10], which are both from Canadian Institute of Cybersecurity, have the raw PCAP files that captured from network. Besides, CIC-IDS2017 has also the structured data that are concluded from PCAP files. CIC-IDS2017 is the dataset that this article uses, but I use the data that are newly extracted from PCAP files in my own way, instead of directly using the structured data given in the original dataset. Details of extracting data will be introduced in IV. Data Gathered.

# Slected Machine Learning Methods

## Decision Tree

Decision trees [11] are a non-parametric, which means the learning changes with the change of the fact (dataset) and has week prior, supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. As mentioned before, the question that this article solves is in essence a binary classification problem, so classification tree is used.

The learning result, or the classifier that the decision tree algorithm outputs is exact a tree-structured model. A decision tree contains a root node, several internal nodes and several nodes, where:

* Each non-leaf node represents a test on a feature attribute.
* Each branch represents the output of this feature attribute on a certain range.
* Each leaf node represents a specific category (classification result).

The process of using a decision tree to make decisions is to start from the root node, test the corresponding feature attributes in the items to be classified, and select the output branch according to its value until it reaches the leaf node. Finally, the category that the leaf node represents is the decision result.

The key step in constructing a decision tree is attributes splitting. Attributes splitting is to construct different branches at a certain node according to the different divisions of a feature, and the goal is to make the items that are classified in each split subset belong to the same category. There are some attribute splitting metrics, or splitting criterions: Information Gain (ID3 algorithm, Iterative Dichotomiser 3), Information Gain Ratio (C4.5 algorithm) and Gini Index (CART algorithm, Classification and Regression Tree).

In this article, the decision tree based on Gini index will be used. Here are the details about the splitting scheme about the tree on Gini index.

In the classification problem, assuming that there are K categories and the probability of the k-th category is pk, the expression of the Gini coefficient is:

Specifically, if the problem is binary classification problem, the expression of the Gini coefficient is:

For sample D, if D is divided into two parts D1 and D2 according to a certain value of feature A, then under the condition of feature A, the Gini index expression of D is:

For classification tree, the CART algorithm chooses the splitting scheme where the attribute minimizes child nodes’ Gini Index. That is, to choose an attribute as the splitting node that minimizes:

## Logistic Regression

Logistic regression [12], despite its name, is a linear model for classification rather than regression. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function. The essence of logistic regression is to assume that the data follows this distribution, and then use maximum likelihood estimation to estimate parameters.

When doing classification, the expected result should be a probability or a set of probabilities which indicates how possible the sample belongs to a specific category. For probability, it has a range of [0, 1]. Normally, a linear regression cannot constantly output a value within [0, 1], but we can put the linear regression expression into another function and now the combined function can always output [0, 1]. In logistic regression, the ‘another’ function is sigmoid function:

Let p mean the output probability. Substitute the linear regression expression into t:

Let’s y is the result of a binary classification that 1 means one category and 0 means another, then:

Now the principle formula of how logistic regression predicts is clarified. The question is, with given a sample dataset x and its corresponding classification results y, how to find parameters , which makes logistic regression can well fit the data.

Usually, to train a machine learning model and make it fit the data, the aim is to minimize the loss function. In logistic regression, the loss function is cross-entropy function:

And the cross-entropy function without is exactly the likelihood function of logistic regression model. That is, minimizing the loss function is equivalent to maximizing the likelihood function. Stochastic gradient descent method and newton’s method can be used to find the solution. Besides, to avoid overfitting, regularization can be added to logistic regression. In this article, regularization is used.

# Data Gathering

As mentioned before, dataset CIC-IDS2017 [10] is used in this article and only port scanning is concerned. In this dataset, port scanning happened at afternoon, Friday, July 7, 2017, so I only downloaded the raw PCAP file named Friday-WorkingHours.pcap. This single file’s size has reached 8.2G. And total PCAP files (5 days record) in this dataset have the size of more than 50G. The too large size in total, which leads to difficulties in downloading, storing in the disk and reading by program, is one reason that I cannot consider all malicious behavior types but only port scanning in this dataset.

In addition, the structured data in CIC-IDS2017, which are concluded from PCAP files, are also needed, because the structured data are labeled while the raw PCAP file have no labels. We cannot know which behavior record in the PCAP file is port scanning behavior without the structured data’s labels. The file name of the needed structured data in dataset is GeneratedLabelledFlows.zip.

## Confirm Port Scanning PCAP Records and Filter Raw PCAP with Time Range

To train a classification model, we must use supervised learning method on labeled data. Therefore, we cannot directly train the model based on unlabeled raw PCAP data. Here I read the structured data of port scanning records (Friday-WorkingHours-Afternoon-PortScan.pcap\_ISCX.csv) given in CIC-IDS2017 and found that the source IP of all entries (rows) which are labeled with ‘PortScan’ is 172.16.0.1. And the rest are all normal behaviors entries and are labeled with ‘Benign’. Here is what the first entry that is a port scanning behavior looks like in the csv file (some of the fields):

1. The First ‘PortScan’ Entry in Structured Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Source IP | … | Timestamp | … | Label |
| 172.16.0.1 | … | 7/7/2017 1:05 | … | PortScan |

Moreover, the timestamp of the last entry that is a port scanning behavior is ‘7/7/2017 3:23’. We can conclude that the port scanning behaviors are all within ‘7/7/2017 1:05’ and ‘7/7/2017 3:23’. To reduce the size of data, I decided to filter the raw PCAP file with the time range 7/7/2017 1:05 to 7/7/2017 3:23. This also reduces the interference of excessive ‘Benign’ data. Even in the filtered and processed data, there are 1487956 ‘Benign’ entries and only 162657 ‘PortScan’ entries.

However, the time range 7/7/2017 1:05 to 7/7/2017 3:23 cannot be directly used to filter the raw PCAP data. First, as stated, this is an ‘afternoon’ data, so the time in 24-hour clock should be 13:05 and 15:23. Second, the timestamp in the structured data is in Canada time zone but the time display format in Wireshark can only be set to UTC time or local time (UTC+8 in Singapore).

To find out the correct time range that can be applied to filter raw PCAP data in Wireshark, here I set Wireshark time display format to UTC time:

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

1. Set Wireshark Time Display Format to UTC Time

Then I found that Canadian time can be from UTC-3 to UTC-7 in summertime (because the date 7/7 is in the summertime period), so I decided to use Canada time 13:05 to compute the corresponding UTC time, from the difference 3 to the difference 7, and compare the very first PCAP entry whose Source IP is 172.16.0.1 with the first ‘PortScan’ entry in structured data to see whether they are consistent.

Finally, I confirmed that the Canada time 13:05 in the structured data is in time zone UTC-3, which means the time range that should be applied to Wireshark to filter the PCAP data in UTC time is 16:05 to 18:23. Specifically, the time of the very first entry where 172.16.0.1 appears as the Source IP is 16:05:33.754382.

I applied the time range 16:05:33.754382 to 18:24 to the raw PCAP file, Friday-WorkingHours.pcap (8.2G) and save the filtered PCAP entries as another file, whose size is only 900M. The condition expression of this time range in Wireshark filter is “frame.time >= "2017-07-07 16:05:33.754382" && frame.time <= "2017-07-07 18:24:00.0"”.

表格

描述已自动生成

1. Apply the Time Range Condition to Filter PCAP Data

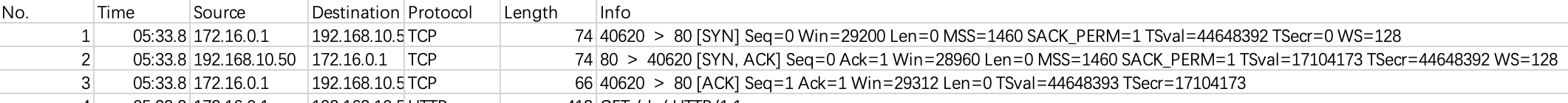
## Generate Structured Data (.csv) from PCAP Data (.pcap)

So far, I have gathered a more accurate raw PCAP file than the original one, but I still have to get structured data to train machine learning models, decision tree and logistic regression. Initially, I try to use Wireshark to open the PCAP file and export the packet dissections as a .csv file. However, the data in exported .csv file are not structured and detailed enough. It leaves unstructured information in the text of ‘Info’ column and has not enough details of the packet.

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

1. Use Wireshark to Export Packet Dissections as CSV File



1. The Example of the Exported CSV Data by Wireshark

According to Shanto Roy’s blog [13], the right way to gather the detailed and structured CSV data from PCAP file is using ‘tshark’ command. By defining the needed fields in ‘tshark’ command, one can gather whatever packets’ information he/she wants from the PCAP file.

Usually, ‘tshark’ tool is included in the installed Wireshark. As a Mac user, first I set a soft link, in order to directly use ‘tshark’ in Terminal:

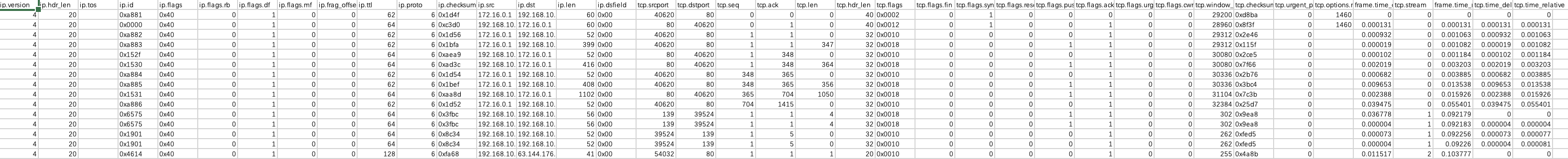
ln -s /Applications/Wireshark.app/Contents/MacOS/tshark /usr/local/bin/tshark

Initially, I had no idea with which kinds of information are exactly needed to train an accurate model, so I referred to Shanto Roy’s practice as well as Tao Liu’s practice and I extracted all the fields used by both. Here is my ‘tshark’ command to extract data from previous PCAP file (the previous PCAP file is named PortScan\_Begin\_to\_End.pcap and the CSV file to generate is named PortScan\_Begin\_to\_End\_tshark\_deltail.csv):

tshark -r PortScan\_Begin\_to\_End.pcap -T fields -E header=y -E separator=, -E quote=d -E occurrence=f -e ip.version -e ip.hdr\_len -e ip.tos -e ip.id -e ip.flags -e ip.flags.rb -e ip.flags.df -e ip.flags.mf -e ip.frag\_offset -e ip.ttl -e ip.proto -e ip.checksum -e ip.src -e ip.dst -e ip.len -e ip.dsfield -e tcp.srcport -e tcp.dstport -e tcp.seq -e tcp.ack -e tcp.len -e tcp.hdr\_len -e tcp.flags -e tcp.flags.fin -e tcp.flags.syn -e tcp.flags.reset -e tcp.flags.push -e tcp.flags.ack -e tcp.flags.urg -e tcp.flags.cwr -e tcp.window\_size -e tcp.checksum -e tcp.urgent\_pointer -e tcp.options.mss\_val -e frame.time\_delta -e tcp.stream -e frame.time\_relative -e tcp.time\_delta -e tcp.time\_relative > PortScan\_Begin\_to\_End\_tshark\_deltail.csv

These commands can also be found in Appendix – A. pcap\_to\_csv.sh.

After running this command in Terminal, I got the .csv file with detailed and structured data in it. Here the figure may be not clear enough. Compared to the data exported via Wireshark that only has 5 fields (without No.), the data exported by ‘tshark’ command has 39 fields.



1. The Example of the Exported CSV Data by ‘tshark’ Command

# Data Preprocessing

Now I have a new dataset in .csv format, which is much more suitable to be used to train machine learning model than other data before. However, the preprocessing of current data is still necessary before I use it to train models.

The full code of preprocessing is in Appendix – B. preprocess\_pcap\_csv.py.

First, use pandas.read\_csv() to read the .csv file as a pandas.DataFrame in program. Then I check the data type (pandas.DataFrame.info())and the situation of the missing value of each field (pandas.DataFrame.isnull().sum()). Here are the results:

1. Field Name, Data Type and Total Missing Valuue Amount

|  |  |  |
| --- | --- | --- |
| Field | Data Type | Total Missing Value Amount |
| ip.version | float64 | 14792 |
| ip.hdr\_len | float64 | 21690 |
| ip.tos | float64 | 1885472 |
| ip.id | object | 21690 |
| ip.flags | object | 21690 |
| ip.flags.rb | float64 | 21690 |
| ip.flags.df | float64 | 21690 |
| ip.flags.mf | float64 | 21690 |
| ip.frag\_offset | float64 | 21690 |
| ip.ttl | float64 | 21690 |
| ip.proto | float64 | 21690 |
| ip.checksum | object | 21690 |
| ip.src | object | 21690 |
| ip.dst | object | 21690 |
| ip.len | float64 | 21690 |
| ip.dsfield | object | 21690 |
| tcp.srcport | float64 | 234853 |
| tcp.dstport | float64 | 234853 |
| tcp.seq | float64 | 234853 |
| tcp.ack | float64 | 234859 |
| tcp.len | float64 | 234989 |
| tcp.hdr\_len | float64 | 234859 |
| tcp.flags | object | 234859 |
| tcp.flags.fin | float64 | 234859 |
| tcp.flags.syn | float64 | 234859 |
| tcp.flags.reset | float64 | 234859 |
| tcp.flags.push | float64 | 234859 |
| tcp.flags.ack | float64 | 234859 |
| tcp.flags.urg | float64 | 234859 |
| tcp.flags.cwr | float64 | 234859 |
| tcp.window\_size | float64 | 234859 |
| tcp.checksum | object | 234859 |
| tcp.urgent\_pointer | float64 | 234859 |
| tcp.options.mss\_val | float64 | 1663957 |
| frame.time\_delta | float64 | 0 |
| tcp.stream | float64 | 234859 |
| frame.time\_relative | float64 | 0 |
| tcp.time\_delta | float64 | 234859 |
| tcp.time\_relative | float64 | 234859 |

Next, I did the following operations:

### pandas.DataFrame.drop\_duplicates(): Drop the rows that are identical. Only keep one of them.

### pandas.DataFrame.dropna(axis = 0, subset = ["ip.src"]): Drop the rows whose ‘ip.src’ field is missing.

### pandas.DataFrame.dropna(axis = 0, subset = ["tcp.window\_size"]): Drop the rows whose ‘ip.src’ field is missing.

### pandas.DataFrame.drop(columns=["ip.tos"]): Drop the column of field ‘ip.tos’, whose values are all mssing.

### pandas.DataFrame.fillna(0, inplace=True): Fill all cells that misses value with 0.

### pandas.DataFrame.applymap(hexadecimal\_to\_decimal): Change hexadecimal number to decimal number. The fields whose value is hexadecimal number are: ‘ip.id’, ‘ip.checksum’, ‘ip.dsfield’, ‘tcp.flags’ and ‘tcp.checksum’. The specific hexadecimal\_to\_decimal funcrtion can be found in Appendix – B. preprocess\_pcap\_csv.py.

### pandas.DataFrame.apply(lambda x: 1 if x["ip.src"] == "172.16.0.1" else 0, axis=1): Add the ‘label’ column. The row whose ‘ip.src’ value is ‘172.16.0.1’ will be assigned the label 1, which means this row represents a port scanning behavior, as illustrated in VI. Data Gathering – A. Conirm Port Scanning PCAP Records and Filter Raw PCAP with Time Range. Otherwise, the label is 0.

### pandas.DataFrame.to\_csv(): Save the processed data as a new file.

# Model Training and Evaluation

After V. Data Preprocessing, there is a CSV file which has the packet information rows and the corresponding labels that indicate whether a packet information row is a port scanning behavior. The packet information rows are the input values to the machine learning model, in other words, they are independent variables or features that the prediction or classification is based on. The labels of these packet information rows are the output values of the model, which are dependent variables and the results of classification (which category that a row having this information belongs to).

In this article I use scikit-learn, a free software machine learning library for the Python to train the models. For both decision tree and logistic regression, the training process of scikit-learn is: first instantiate a class of the model, then call the model.fit(X, Y) method to train the model, where X is feature array and Y is label array. The feature array and the label array should correspond to each other row by row.

Also, the train\_test\_split() method in scikit-learn can split the origin X and Y into training set (train\_X and train\_Y) and validation test (test\_X and test\_Y). That is, we can use train\_X and train\_Y to train the model then use test\_X and test\_Y to evaluate the accuracy of the trained model.

For evaluation, call the method model.score(test\_X, test\_Y) of a trained model and the accuracy of the model on this test set will be output.

Therefore, what I should do first in the code is to transform the CSV data to X (feature array) and Y (label array), then split them to put train\_X, train\_Y and test\_X, test\_Y . Call model.fit(train\_X, train\_Y) to train the model and call model.score(test\_X, test\_Y).

The full code of decision tree is in Appendix – C. decision\_tree.py. Then code of logistic regression is Appendix – D. logistic\_regression.py. Here I will not give the explanation of code line by line.

2 things that should be mentioned in my training process:

* Feature array X doesn’t include column ‘ip.src’ and column ‘ip.dst’. The reason to drop them is that they are just a kind of identification, like that we cannot predict whether a person will do something based on his or her ID number.
* When using train\_test\_split() method to split train and test data, parameter ‘stratify = array\_Y’ must be sent into the method. This is because the difference between the number of positive and negative samples is too large. Without stratified split, the test data whose size is small may not contain any positive sample.

The decision tree model’s tree structure is very readable. After training decision tree model, I visualize its tree structure. The code of tree visualization is at the end of Appendix – C. decision\_tree.py. Also, a higher resolution tree figure is put in Appendix – E. tree\_graph.png.

日程表

描述已自动生成

1. Tree Visualization (Higher Resolution Image in Appendix – E. tree\_graph.png)

As introduced before, decision tree uses Gini index to choose which feature should be used to split nodes. Therefore, each feature has its feature importance in the trained tree. Here I give the table of each feature’s importance that are greater than and visualize them. Those feature importance that is not listed here is all 0.

The code of feature importance visualization is in Appendix – F. draw\_importances.py.

1. Feature Importance That is Greater than 0

|  |  |
| --- | --- |
| Feature | Importance |
| ip.ttl | 0.18392902 |
| tcp.srcport | 0.00210637 |
| tcp.dstport | 0.00140748 |
| tcp.ack | 0.00124951 |
| tcp.checksum | 0.00001135 |
| tcp.stream | 0.00016605 |
| tcp.time\_delta | 0.00001714 |
| tcp.time\_relative | 0.81111307 |

图表

描述已自动生成

1. Feature Importance That is Greater than 0

For logistic regression, the model training process is to find value of the and in 1 / ( . Here I get the value of and from the trained model. is a matrix that each its value represents the corresponding feature’s coefficient in the trained logistic regression model and is the intercept.

1. Feature and Its Coefficient Value (Value in )

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Coefficient | Feature | Coefficient |
| ip.version | -0.0021606 | tcp.hdr\_len | 0.01986365 |
| ip.hdr\_len | -0.0108031 | tcp.flags | -0.0563767 |
| ip.id | 5.74E-06 | tcp.flags.fin | -4.07E-05 |
| ip.flags | -0.0262486 | tcp.flags.syn | 0.00156534 |
| ip.flags.rb | 0 | tcp.flags.reset | -0.0010277 |
| ip.flags.df | -0.0004101 | tcp.flags.push | -4.11E-05 |
| ip.flags.mf | 0 | tcp.flags.ack | -0.003427 |
| ip.frag\_offset | 0 | tcp.flags.urg | 0 |
| ip.ttl | -0.1168252 | tcp.flags.cwr | -9.93E-07 |
| ip.proto | -0.0032409 | tcp.window\_size | -1.84E-05 |
| ip.checksum | -2.81E-06 | tcp.checksum | -1.50E-06 |
| ip.len | 0.00230942 | tcp.urgent\_pointer | -0.0002243 |
| ip.dsfield | -0.004075 | tcp.options.mss\_val | 0.00155511 |
| tcp.srcport | 8.78E-05 | frame.time\_delta | -2.90E-05 |
| tcp.dstport | -2.85E-05 | tcp.stream | 4.33E-06 |
| tcp.seq | -0.0188501 | frame.time\_relative | 0.0006402 |
| tcp.ack | -2.56E-08 | tcp.time\_delta | -0.0007397 |
| tcp.len | -0.0067512 | tcp.time\_relative | -0.0031623 |

And the (intercept) is -0.00054016.

Here are the performance of decision tree model and logistic regression model. The accuracy is returned by model.score(test\_X, test\_Y) and the training time is the difference of time.time() before and after the execution of model.fit(train\_X. train\_Y).

1. Model’s Performance

|  |  |  |
| --- | --- | --- |
| Model/Performance | Accuracy | Training Time |
| Decision Tree | 0.99998 | 3.00903s |
| Logistic Regression | 0.99242 | 35.33352s |

We can see that decision tree model is more accurate on the validation set and the training time of decision tree model is much less than logistic regression.

Moreover, all the values (feature importance, coefficient and accuracy) produced by training and testing may slightly fluctuate each time the code is run, because there is random state in training. The change will not be large overall.

# Test Model on Given Test Set

[To do]

# Conclusion

Raw PCAP file has no label. Even the existed dataset needs us to compare its structured data and its PCAP file carefully to confirm which label should be given to a packet information row of the PCAP file. The timestamp and the IP addresses will help a lot for the confirmation processes.

Timestamp can be used to filter the records in the PCAP file, which can largely reduce the size of data. Finally, use ‘tshark’ command line tool to save the detailed information in the PCAP file as CSV file. The CSV file with detailed information is suitable to be used to train machine learning model.

Handle missing values and drop those columns that are meaningless features to the model before using data to train the model. Feature importance and tree structure can be gathered from scikit-learn’s decision tree model instance. Coefficient and intercept can be gathered from logistic regression model instance.

Decision tree model’s training is much faster than logistic regression model. The accuracy of decision tree model is also little better.

##### Acknowledgment

I am very grateful to Professor Jonathan Pan. Prof. Pan is meticulous and patient. He is very friendly to students. The knowledge he teaches in the course is also very useful.

##### References

1. Avast Business. (n.d.). What is port scanning and how does it work? | Avast. Avast Antivirus. Retrieved March 16, 2022, from https://www.avast.com/en-sg/business/resources/what-is-port-scanning.
2. Denning, D. E. (1987). An intrusion-detection model. IEEE Transactions on software engineering, (2), 222-232.
3. Liao, Y., & Vemuri, V. R. (2002). Use of k-nearest neighbor classifier for intrusion detection. Computers & security, 21(5), 439-448.
4. Chen, W. H., Hsu, S. H., & Shen, H. P. (2005). Application of SVM and ANN for intrusion detection. Computers & Operations Research, 32(10), 2617-2634.
5. Farnaaz, N., & Jabbar, M. A. (2016). Random forest modeling for network intrusion detection system. Procedia Computer Science, 89, 213-217.
6. A.I.L. (2021, September 16). Detection of abnormal network traffic through deep learning. CSDN. Retrieved March 16, 2022, from https://blog.csdn.net/weixin\_39032619/article/details/120337693
7. KDD Cup. (1999, October 28). KDD Cup 1999 Data. KDD. Retrieved March 17, 2022, from http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
8. Stratosphere Laboratory. (n.d.). The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background traffic. Stratosphere IPS. Retrieved March 17, 2022, from https://www.stratosphereips.org/datasets-ctu13/
9. Canadian Institute for Cybersecurity. (2012, May). IDS 2012 | Datasets | Research | Canadian Institute for Cybersecurity | UNB. Retrieved March 18, 2022, from https://www.unb.ca/cic/datasets/ids.html
10. Canadian Institute for Cybersecurity. (2017, July). IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB. Retrieved March 18, 2022, from https://www.unb.ca/cic/datasets/ids-2017.html
11. Scikit-learn Developers. (n.d.). 1.10. Decision Trees. Scikit-Learn. Retrieved March 18, 2022, from https://scikit-learn.org/stable/modules/tree.html#tree
12. Scikit-learn Developers. (n.d.). 1.1. Linear Models. Scikit-Learn. Retrieved March 20, 2022, from https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
13. Roy, S. (2021, February 5). *Convert PCAP files to CSV for Network Traffic Analysis*. Roy’s Blog. Retrieved March 23, 2022, from https://shantoroy.com/networking/convert-pcap-to-csv-using-tshark/
14. Liu, T. (2020, June 1). *Build a Machine Learning Model for Network Flow Classification*. LinkedIn. Retrieved March 23, 2022, from https://www.linkedin.com/pulse/build-machine-learning-model-network-flow-tao-liu/

##### Appendix

## pcap\_to\_csv.sh

*# soft link tshark*

ln -s /Applications/Wireshark.app/Contents/MacOS/tshark /usr/local/bin/tshark

*# generate .csv*

tshark -r PortScan\_Begin\_to\_End.pcap -T fields -E header=y -E separator=, -E quote=d -E occurrence=f -e ip.version -e ip.hdr\_len -e ip.tos -e ip.id -e ip.flags -e ip.flags.rb -e ip.flags.df -e ip.flags.mf -e ip.frag\_offset -e ip.ttl -e ip.proto -e ip.checksum -e ip.src -e ip.dst -e ip.len -e ip.dsfield -e tcp.srcport -e tcp.dstport -e tcp.seq -e tcp.ack -e tcp.len -e tcp.hdr\_len -e tcp.flags -e tcp.flags.fin -e tcp.flags.syn -e tcp.flags.reset -e tcp.flags.push -e tcp.flags.ack -e tcp.flags.urg -e tcp.flags.cwr -e tcp.window\_size -e tcp.checksum -e tcp.urgent\_pointer -e tcp.options.mss\_val -e frame.time\_delta -e tcp.stream -e frame.time\_relative -e tcp.time\_delta -e tcp.time\_relative > PortScan\_Begin\_to\_End\_tshark\_deltail.csv

## preprocess\_pcap\_csv.py

import pandas as pd

*#pd.set\_option('displ')*

pcap\_detail\_df = pd.read\_csv("PortScan\_Begin\_to\_End\_tshark\_deltail.csv")

pcap\_detail\_df.info() *# 直接查看信息*

'''

Data columns (total 39 columns):

# Column Dtype

--- ------ -----

0 ip.version float64

1 ip.hdr\_len float64

2 ip.tos float64

3 ip.id object

4 ip.flags object

5 ip.flags.rb float64

6 ip.flags.df float64

7 ip.flags.mf float64

8 ip.frag\_offset float64

9 ip.ttl float64

10 ip.proto float64

11 ip.checksum object

12 ip.src object

13 ip.dst object

14 ip.len float64

15 ip.dsfield object

16 tcp.srcport float64

17 tcp.dstport float64

18 tcp.seq float64

19 tcp.ack float64

20 tcp.len float64

21 tcp.hdr\_len float64

22 tcp.flags object

23 tcp.flags.fin float64

24 tcp.flags.syn float64

25 tcp.flags.reset float64

26 tcp.flags.push float64

27 tcp.flags.ack float64

28 tcp.flags.urg float64

29 tcp.flags.cwr float64

30 tcp.window\_size float64

31 tcp.checksum object

32 tcp.urgent\_pointer float64

33 tcp.options.mss\_val float64

34 frame.time\_delta float64

35 tcp.stream float64

36 frame.time\_relative float64

37 tcp.time\_delta float64

38 tcp.time\_relative float64

dtypes: float64(31), object(8)

memory usage: 561.0+ MB

'''

print(pcap\_detail\_df.isnull().sum()) *# 每列有多少个缺失值*

'''

memory usage: 561.0+ MB

ip.version 14792

ip.hdr\_len 21690

ip.tos 1885472

ip.id 21690

ip.flags 21690

ip.flags.rb 21690

ip.flags.df 21690

ip.flags.mf 21690

ip.frag\_offset 21690

ip.ttl 21690

ip.proto 21690

ip.checksum 21690

ip.src 21690

ip.dst 21690

ip.len 21690

ip.dsfield 21690

tcp.srcport 234853

tcp.dstport 234853

tcp.seq 234853

tcp.ack 234859

tcp.len 234989

tcp.hdr\_len 234859

tcp.flags 234859

tcp.flags.fin 234859

tcp.flags.syn 234859

tcp.flags.reset 234859

tcp.flags.push 234859

tcp.flags.ack 234859

tcp.flags.urg 234859

tcp.flags.cwr 234859

tcp.window\_size 234859

tcp.checksum 234859

tcp.urgent\_pointer 234859

tcp.options.mss\_val 1663957

frame.time\_delta 0

tcp.stream 234859

frame.time\_relative 0

tcp.time\_delta 234859

tcp.time\_relative 234859

'''

print(pcap\_detail\_df.isna().sum(1)) *# 每行有多少个缺失值*

*# drop totally duplicated rows*

pcap\_detail\_df = pcap\_detail\_df.drop\_duplicates()

*# drop the rows contains Nan*

pcap\_detail\_df = pcap\_detail\_df.dropna(axis = 0, subset = ["ip.src"])

pcap\_detail\_df = pcap\_detail\_df.dropna(axis = 0, subset = ["tcp.window\_size"])

*# drop ip.tos, which are all Nan*

pcap\_detail\_df = pcap\_detail\_df.drop(columns=["ip.tos"])

*# delete the quote of elements*

def delete\_quote(x: str):

return x.strip("") if x is str else x

pcap\_detail\_df = pcap\_detail\_df.applymap(delete\_quote)

*# fill Nan wiith 0*

pcap\_detail\_df.fillna(0, inplace=True)

*# change hexadecimal number to decimal number*

hexadecimal\_columns = ["ip.id", "ip.checksum", "ip.dsfield", "tcp.flags", "tcp.checksum"]

def hexadecimal\_to\_decimal(x: str):

if "0x" in str(x):

return int(str(x), 16)

return x

pcap\_detail\_df = pcap\_detail\_df.applymap(hexadecimal\_to\_decimal)

*# add label*

pcap\_detail\_df["label"] = pcap\_detail\_df.apply(lambda x: 1 if x["ip.src"] == "172.16.0.1" else 0, axis=1)

print(pcap\_detail\_df.groupby('label').size())

print(pcap\_detail\_df.head(5))

pcap\_detail\_df.to\_csv("PortScan\_Begin\_to\_End\_tshark\_deltail\_with\_label.csv", index=False)

## decision\_tree.py

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn import tree

dataset = pd.read\_csv("PortScan\_Begin\_to\_End\_tshark\_deltail\_with\_label.csv")

'''

label

0 1487956

1 162657

dtype: int64

'''

print(dataset.groupby('label').size())

label = dataset["label"]

features = dataset.drop(["label", "ip.src", "ip.dst"], axis=1)

array\_X = features.values

print(features.columns)

print(array\_X)

array\_Y = label.values

train\_X, test\_X, train\_Y, test\_Y = train\_test\_split(array\_X, array\_Y, train\_size = 0.2, random\_state = 42, stratify = array\_Y)

clf = tree.DecisionTreeClassifier()

import time

train\_begin = time.time()

clf = clf.fit(train\_X, train\_Y)

print("Total Train Time:", time.time() - train\_begin)

result = clf.score(test\_X, test\_Y)

print(result)

print(clf.feature\_importances\_)

feature\_importances\_datafram = pd.DataFrame({

"Feature": features.columns,

"Importance": clf.feature\_importances\_

})

feature\_importances\_datafram.to\_csv("decision\_tree\_importances.csv", index=False)

import matplotlib.pyplot as plt

from sklearn import tree

plt.figure(dpi=300)

tree.plot\_tree(clf, feature\_names=features.columns,filled=True)

plt.savefig('tree.png',format='png')

plt.show()

'''

# draw tree

'''

import graphviz

tree.plot\_tree(clf)

dot\_data = tree.export\_graphviz(clf, out\_file="tree\_graph.dot",

feature\_names=features.columns,

class\_names=True,

filled=True,

rounded=True,

special\_characters=True)

graph = graphviz.Source(dot\_data)

*#print(graph)*

plt.show()

*# draw\_importance*

## logistic\_regression.py

import pandas as pd

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

dataset = pd.read\_csv("PortScan\_Begin\_to\_End\_tshark\_deltail\_with\_label.csv")

'''

label

0 1487956

1 162657

dtype: int64

'''

print(dataset.groupby('label').size())

label = dataset["label"]

features = dataset.drop(["label", "ip.src", "ip.dst"], axis=1)

array\_X = features.values

print(features.columns)

print(array\_X)

array\_Y = label.values

train\_X, test\_X, train\_Y, test\_Y = train\_test\_split(array\_X, array\_Y, train\_size = 0.2, random\_state = 42, stratify = array\_Y)

clf = LogisticRegression(max\_iter=500)

import time

train\_begin = time.time()

clf.fit(train\_X, train\_Y)

print(clf.score(test\_X, test\_Y))

print("Total Train Time:", time.time() - train\_begin)

print(clf.get\_params())

print(clf.coef\_)

print(clf.intercept\_)

coef\_datafram = pd.DataFrame({

"Feature": features.columns,

"Importance": clf.coef\_[0]

})

coef\_datafram.to\_csv("logistic\_regression\_coef.csv", index=False)

## tree\_graph.png

日程表

描述已自动生成

## draw\_importances.py

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

feature\_importances\_df = pd.read\_csv("decision\_tree\_importances.csv")

feature\_importances\_df = feature\_importances\_df.sort\_values(by=["Importance"], ascending=False)

feature\_list = feature\_importances\_df["Feature"].to\_list()[0: 8]

print(feature\_list)

importance\_array = feature\_importances\_df["Importance"].values[0: 8]

print(importance\_array)

plt.figure(dpi=300)

plt.title("Features and Importances")*#, fontsize=16)*

plt.bar(range(len(importance\_array)), importance\_array)

plt.xticks(range(len(importance\_array)), feature\_list, rotation = 30, size="small")*#, fontsize=12)*

plt.xlabel('Feature')*#, size='larger')*

plt.ylabel('Importance')*#, size='larger')*

plt.grid()

for a,b in zip(range(len(importance\_array)), importance\_array):

plt.text(a, b+0.01, '%.07f' % b, ha='center', va= 'bottom', rotation=30, size="small")*#,fontsize=12)*

plt.savefig('Figure\_1\_1.png',format='png')

plt.show()