**PROJECT REPORT**

**INTRODUCTION**

The malware memory analysis dataset is from the Canadian Institute for Cybersecurity (CIC). The Canadian Institute for Cybersecurity (CIC) is a vast multidisciplinary training, research and development organization that taps the knowledge of academics from the social sciences, business, computer science, engineering, law, and science. Their vision is to help Canada be a global leader in cybersecurity research, innovation, and education (University of New Brunswick, n.d.).

The malware memory analysis dataset contains data about obfuscated malware. Malware that is obfuscated hides to evade discovery and destruction. The goal of this dataset is to evaluate memory-based obfuscated malware detection techniques. In other words, can we detect obfuscated malware using memory data? There are two classes represented in the dataset: *benign* and *malicious*. The malicious records consist of Spyware, Ransomware and Trojan malware. There are *58596* entries and *57* variables in the dataset. The dataset can be downloaded here: [*https://www.unb.ca/cic/datasets/malmem-2022.html*](https://www.unb.ca/cic/datasets/malmem-2022.html).

**STORAGE**

Let’s say we have a company called Malware Hunters. This company helps users detect malware in their systems using Artificial Intelligence. They gather memory data weekly. They have been storing this data locally, but they are considering using a cloud-based system for storage.

Their big data engineer Abiodun suggested using a cloud-based system because storing data in the cloud ensures it is safe and a backup available. The data can be accessed from any location (Thompson, 2022). A cloud-based system is also scalable. As the size of the data increases, they can acquire more space from their cloud service provider (Kochovski, 2022). There are different cloud services you can use. We have Google Cloud Platform (GCP), Amazon Web Service (AWS), Microsoft Azure, IBM Cloud (Kyndryl) etc. Malware Hunters prefer using AWS because of their partnership with Amazon.

To store their data in AWS, they created a bucket using S3. To do this Abiodun logged into the AWS management console. He selected ‘Services’ at the top menu, and under the *Storage* section, he clicked on *S3*. He then clicked on the *‘Create a Bucket’* button. He chose a unique name for the bucket name. The bucket name is ‘obfuscated-malmem-2022’. Next, he had to select the AWS region where the bucket should be located. Abioidun knows that in order to reduce latency and expenses, as well as optimize for speed, he had to choose a region close to Africa. He picked the *Europe (Ireland)* region with the code name *eu-west-1*.

After creating the bucket, he opened it and clicked on the upload button. He then uploaded the ‘Obfuscated-MalMem2022.csv’ file to the bucket. With Amazon S3, Malware Hunter only pay for the storage they use. There are different S3 storage classes. They include S3 standard (used to store active data), S3 standard infrequent access (for storing less active data) and amazon glazier (for storing archive data). Since the data will be accessed frequently, the company uses *S3 Standard* storage class (AWS, n.d)

**DATA WRANGLING**

**Data Gathering**

The dataset was manually downloaded from the website [*https://www.unb.ca/cic/datasets/malmem-2022.html*](https://www.unb.ca/cic/datasets/malmem-2022.html). The dataset is a CSV (Comma-Separated Values) file. It was loaded into the Jupyter Notebook using Pandas.

**Data Assessing**

I observed the following points while assessing the quality and structure of the data.

1. There are no missing values in the dataset.
2. There are 534 duplicate records in the dataset.
3. The *Class* and *Category* variables are of the object data type.
4. Some values in the *Category* variable have strange characters like -00a2c6bab1e53f679cdd4fdc772cd291928c109b9b747652639a1700d844f719-1.raw.
5. There is a structural issue in the *Category* variable. Both categories of attack and type are included in the same column. For example, *Spyware-Gator* and *Ransomware-Ako.*
6. All the values in *the pslist.nprocs64bit, svcscan.interactive\_process\_services, handles.nport, svcscan.interactive\_process\_services* variables are zeros.
7. There are possible outliers in the *pslist.nppid, pslist.avg\_handlers, handles.avg\_handles\_per\_proc, handles.nfile, handles.ndesktop, handles.nkey, handles.nthread, handles.nsemaphore, handles.nsection, malfind.ninjections, malfind.commitCharge, malfind.protection, malfind.uniqueInjections, psxview.not\_in\_pslist, psxview.not\_in\_ethread\_pool, psxview.not\_in\_pspcid\_list, psxview.not\_in\_csrss\_handles, psxview.not\_in\_session, and psxview.not\_in\_deskthrd* variables.

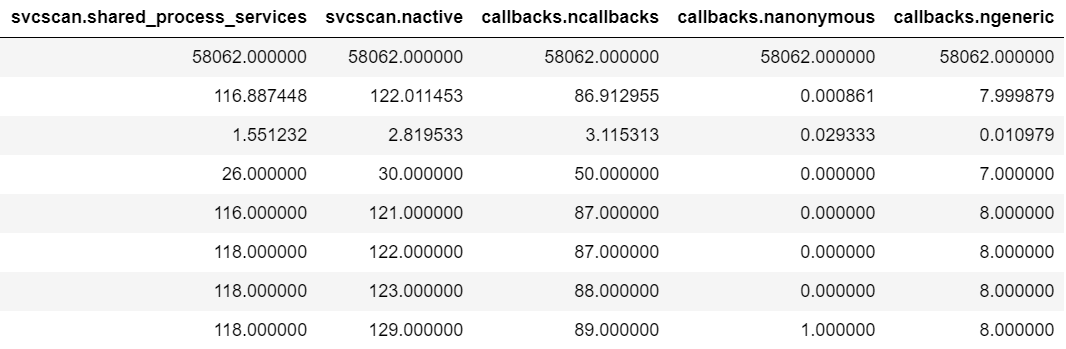
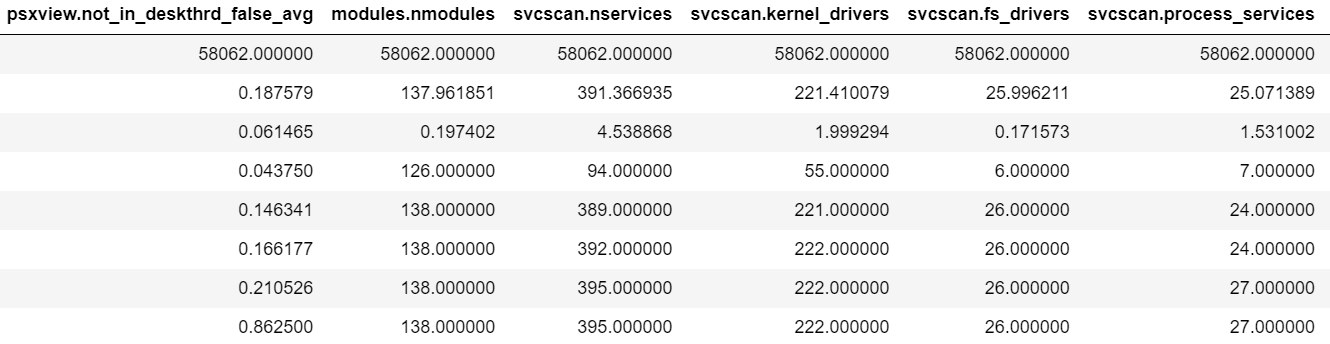
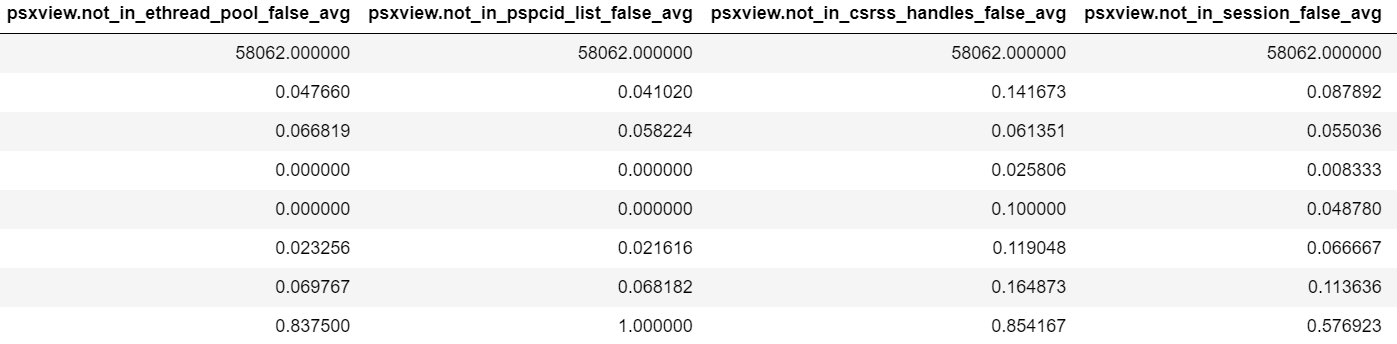
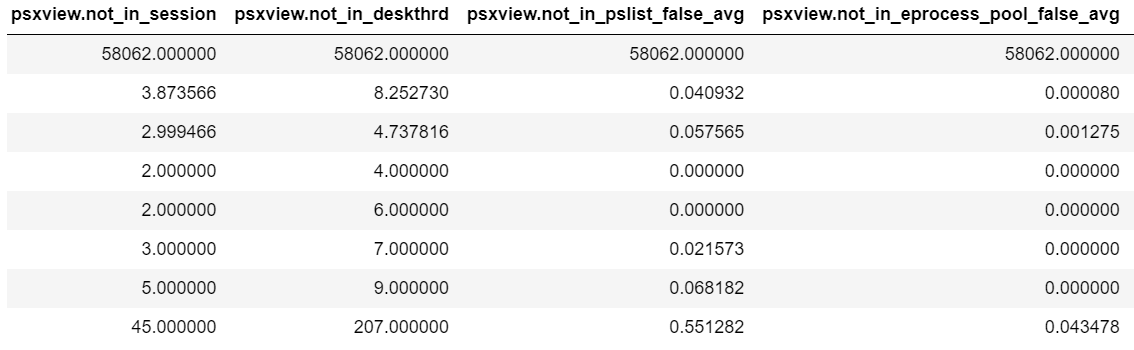
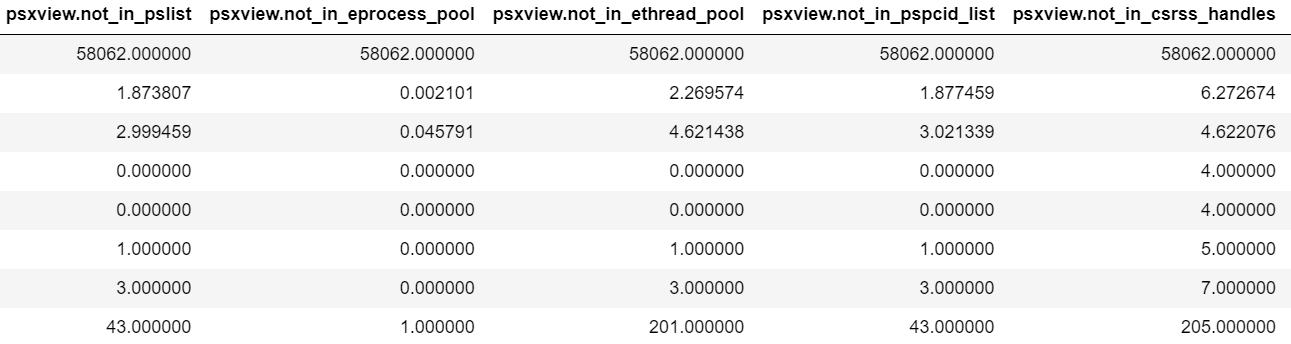
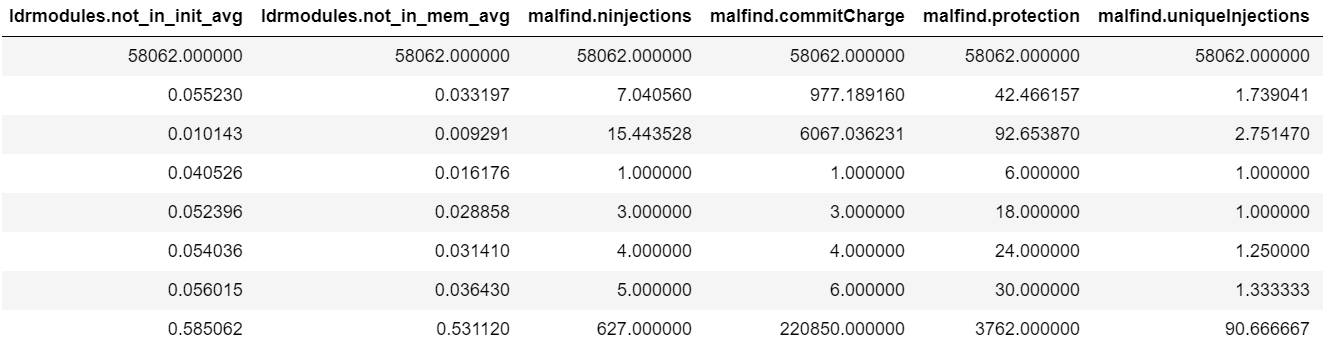
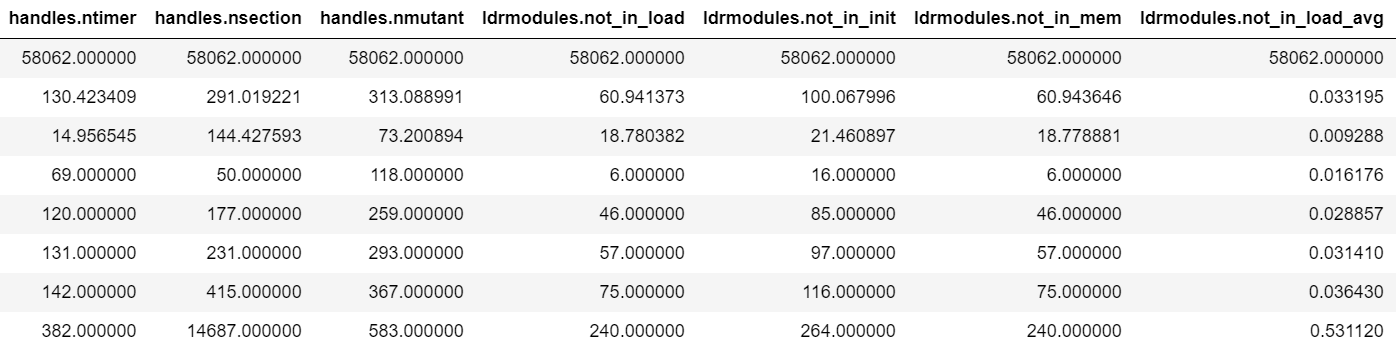
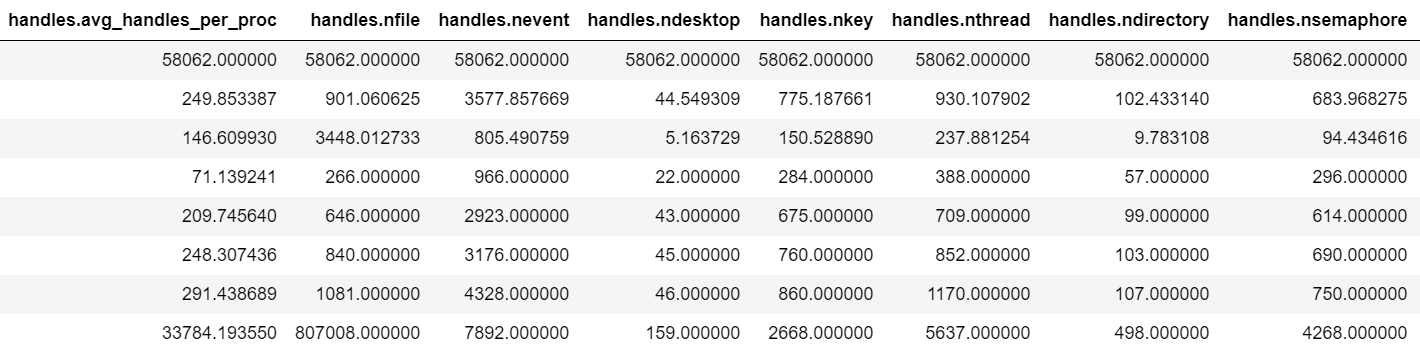
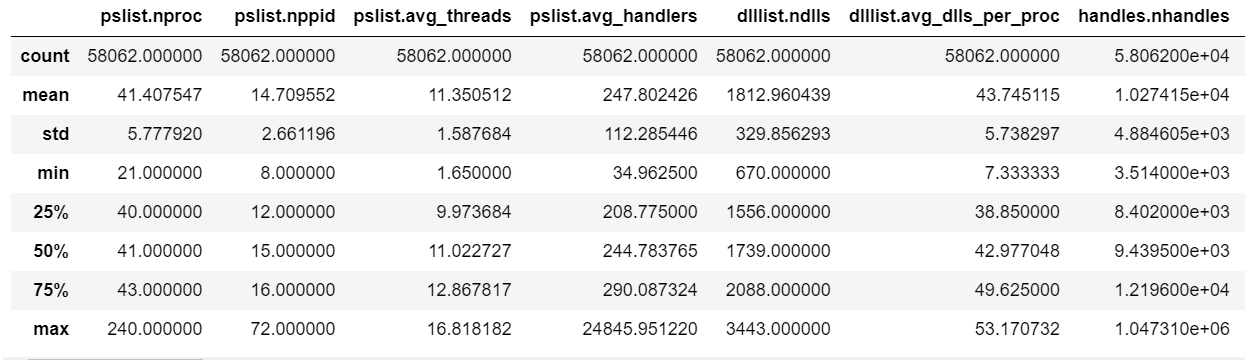
**Data Cleaning**

1. I removed duplicate records.
2. I remove the strange characters in the *Category* variable.
3. I fixed the structural issue in the *Category*column by creating a new column, *type*, to store the family of malware.
4. I removed the *pslist.nprocs64bit, svcscan.interactive\_process\_services, handles.nport, svcscan.interactive\_process\_services* variables.

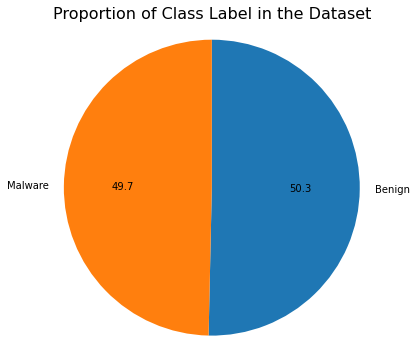
Other data pre-processing steps like normalization will be performed before training a machine learning model on the data.

**DESCRIPTIVE ANALYSIS**

The images below show the summary statistics of the variables in the dataset. The summary statistics includes the count, the mean, the standard deviation, the minimum value, the 25th percentile, the 50th percentile (the median), the 75th percentile and the maximum value for all the variables in the dataset.

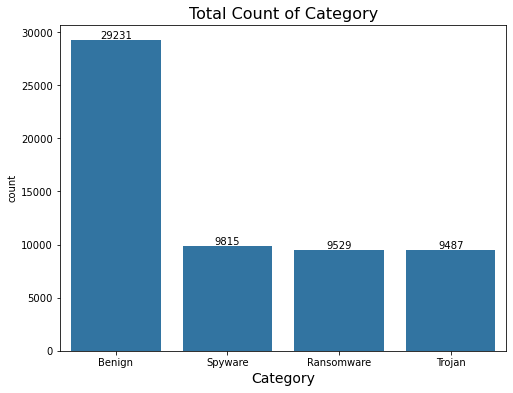


*Figure 1-9: Summary Statistics of the variables in the dataset*



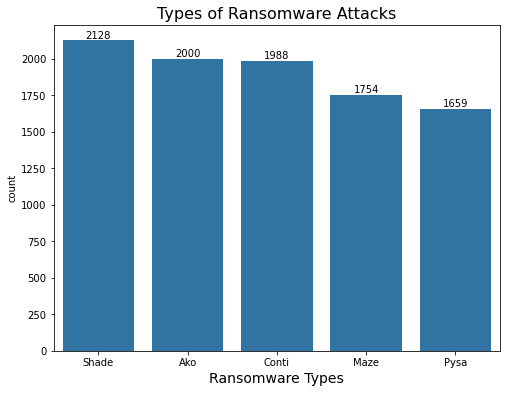
*Figure 10: Proportion of class labels in the dataset*

From the chart above, 49.7% of the data in the dataset are records of *malicious* attacks. While 50.3% of the data are *benign*. The classes in the dataset are balanced.



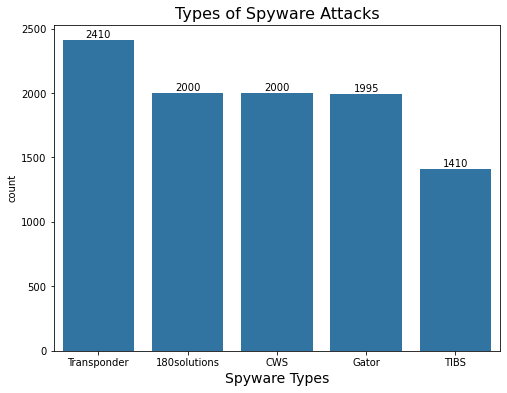
*Figure 11: Proportion of categories in the dataset*

There are *29231* records of benign connections, *9815* records of Spyware attacks, *9529* records of Ransomware attacks and *9487* records of Trojan attacks.



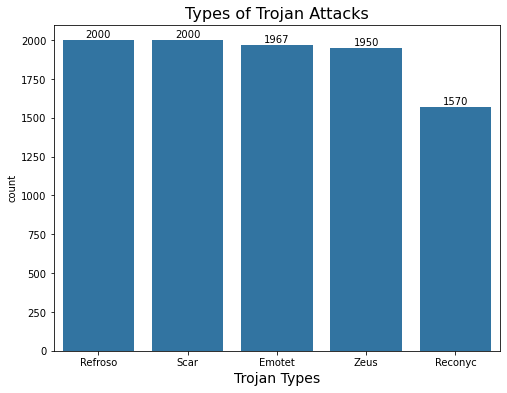
*Figure 12: Types of Ransomware attacks*

There are five types of ransomware attacks in the dataset. These are *shade, ako, conti, maze,*and *pysa*. There are *2128* shade ransomware attacks, *2000* ako ransomware attacks, *1988* conti ransomware attacks, *1754* maze ransomware attacks and *1659* pysa ransomware attacks.



*Figure 13: Types of Spyware attacks*

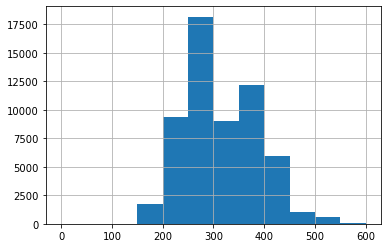
There are five types of spyware attacks in the dataset. These are *Transponder, 180solutions, CWS, Gator,*and *TIBS*. There are *2410* Transponder spyware attacks, *2000* 180solutions spyware attacks, *2000* CWS spyware attacks, *1995* Gator spyware attacks and *1410* TIBS spyware attacks.



*Figure 14: Types of Trojan attacks*

There are five types of Trojan attacks in the dataset. These are *Refroso, Scar, Emotet, Zeus,* and*Reconyc*. There are *2000* Refroso attacks, *2000* Scar attacks, *1967* Emotet attacks, *1950* Zeus attacks and *1570* Reconyc attacks.

**Distribution of *handles.nmutant* variable**

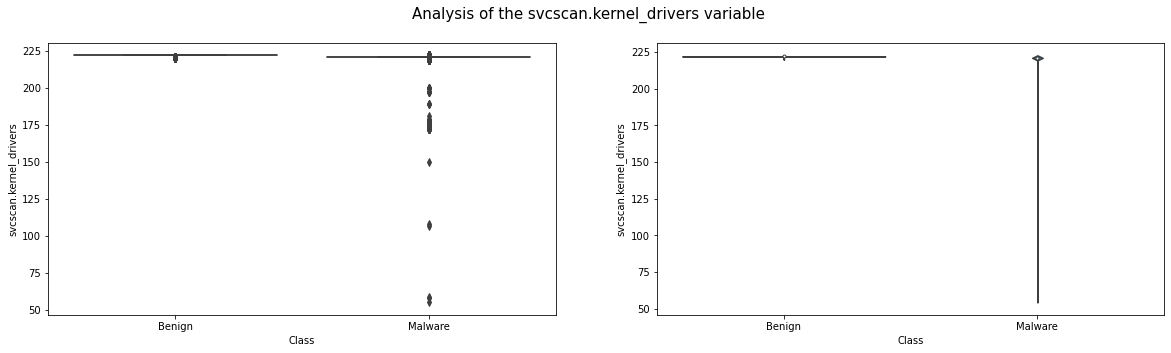


*Figure 15: Distribution of the handles.nmutant variable.*

Most of the values in the handles.nmutant variable are within the range of 250-300.

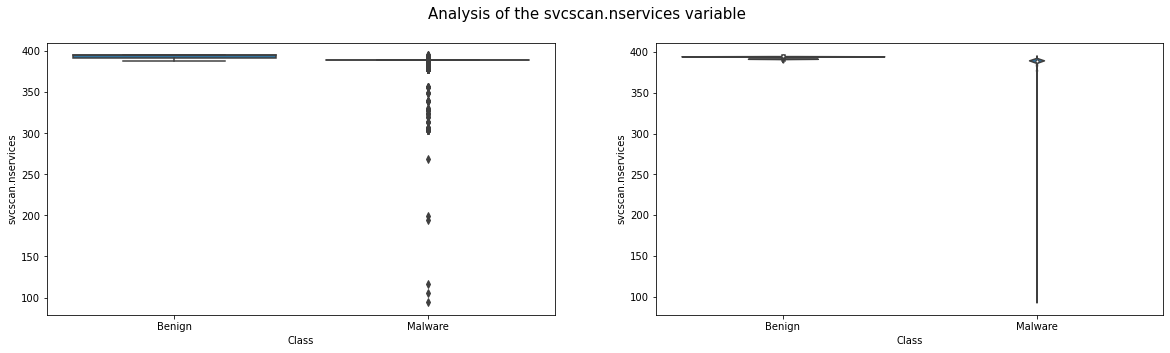
**DIAGNOSTIC ANALYSIS**

In this section, I try to answer the question “why are some memory records malware attacks?”



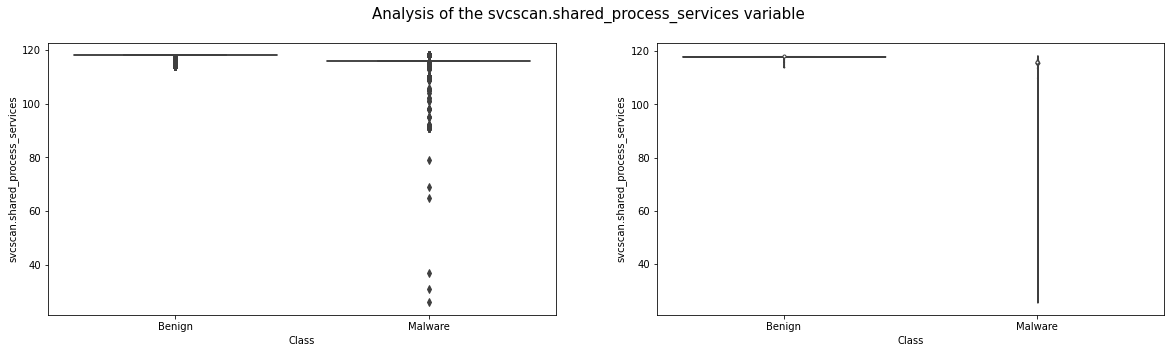
*Figure 16: Analysis of the svcscan.kernel\_drivers variable*

The plot above on the left is a box plot, while the plot on the right is a violin plot. The *svcscan.kernel\_drivers* variable of malware attacks have values less than 200. If the value of the *svcscan.kernel\_drivers* variable is below 200, there is a high chance that it is a malware attack.



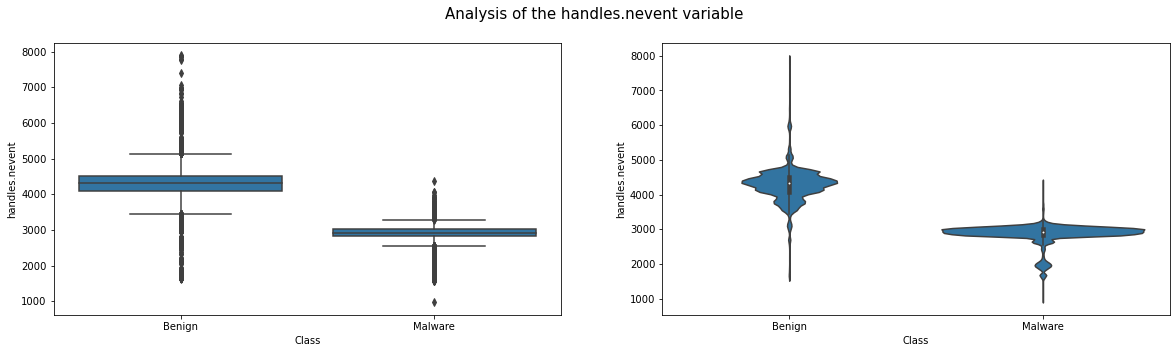
*Figure 17: Analysis of the svcscan.nservices variable*

The svcscan.nservices variable of malware attacks have values less than 350. If the value of the svcscan.nservices variable is below 350, there is a high chance that it is a malware attack.



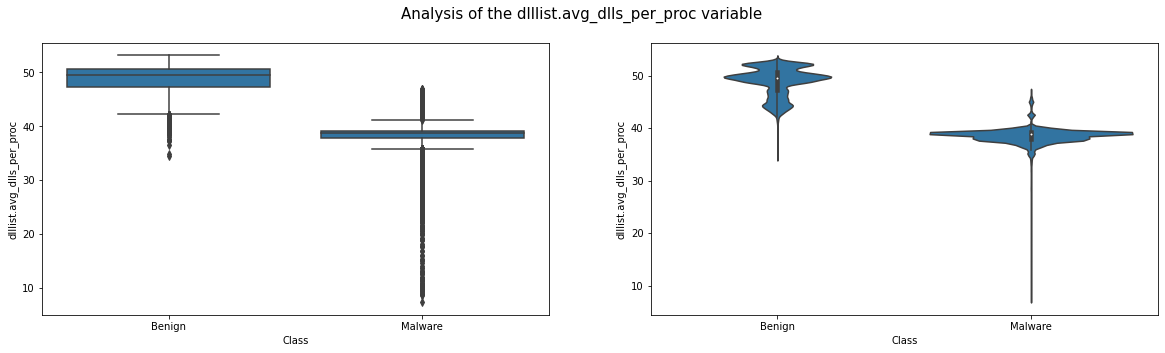
*Figure 18: Analysis of the svcscan.shared\_process\_services variable*

The svcscan.shared\_process\_services variable of malware attacks has values less than 100. If the value of the svcscan.shared\_process\_services variable is below 100, there is a high chance that it is a malware attack.



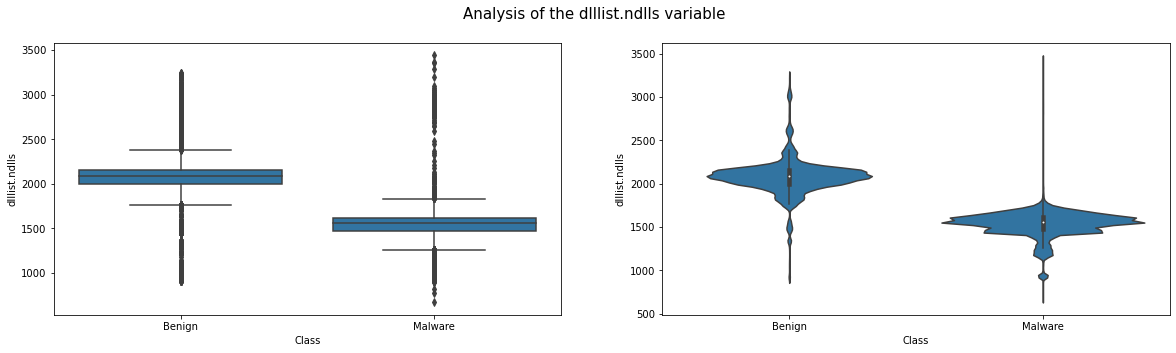
*Figure 19: Analysis of the handles.nevent variable*

For the variable *handles.nevent*, most of the malware attacks have a value of 3000 as indicated in the violin plot above. Most benign records are within the range of 3500-5000. Values below 3000 could be malicious attacks.



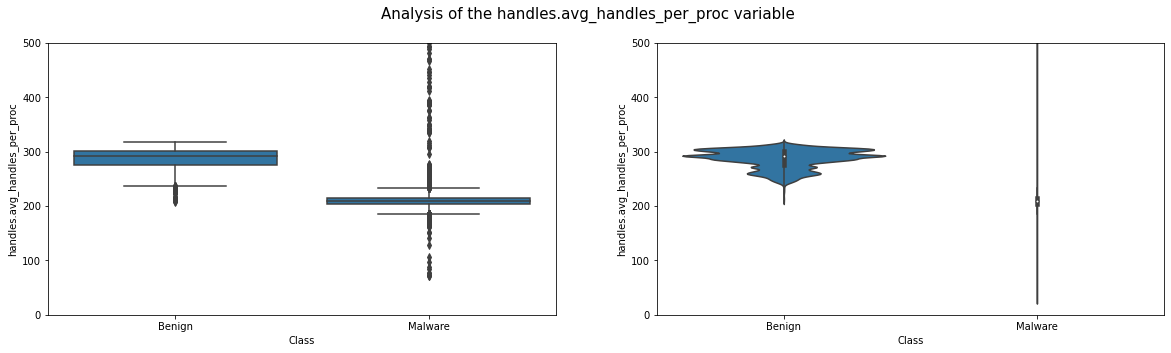
*Figure 20: Analysis of the dlllist.avg\_dlls\_per\_proc* *variable*

For the variable *dlllist.avg\_dlls\_per\_proc*, most of the malware attacks have values ranging from 35 - 40 as indicated in the violin plot above. Most benign records are above the value 40. If the value of *dlllist.avg\_dlls\_per\_proc* is below 40, there is a high chance that it is a malware attack.



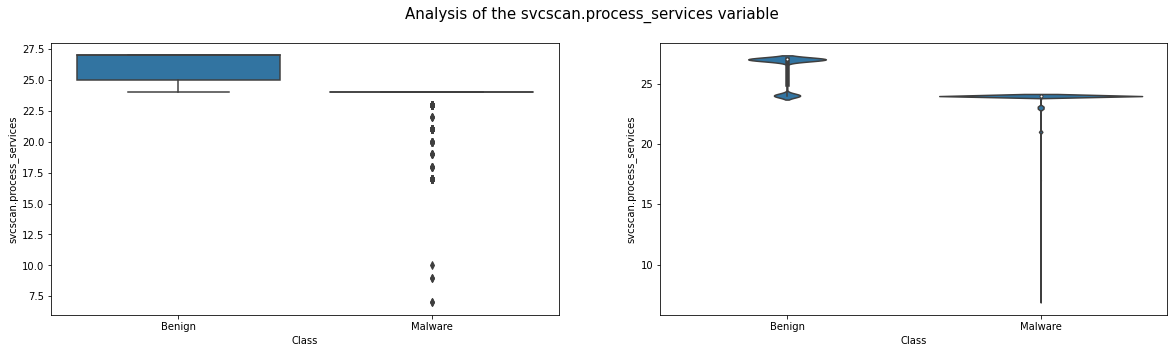
*Figure 21: Analysis of the dlllist.ndlls variable*

For the variable *dlllist.ndlls*, most of the malware attacks have values ranging from 1400 - 1700 as indicated in the violin plot above. Most benign records are within the range of 2000-2300. If the value of *dlllist.ndlls* is below 1500, there is a high chance that it is a malicious attack.



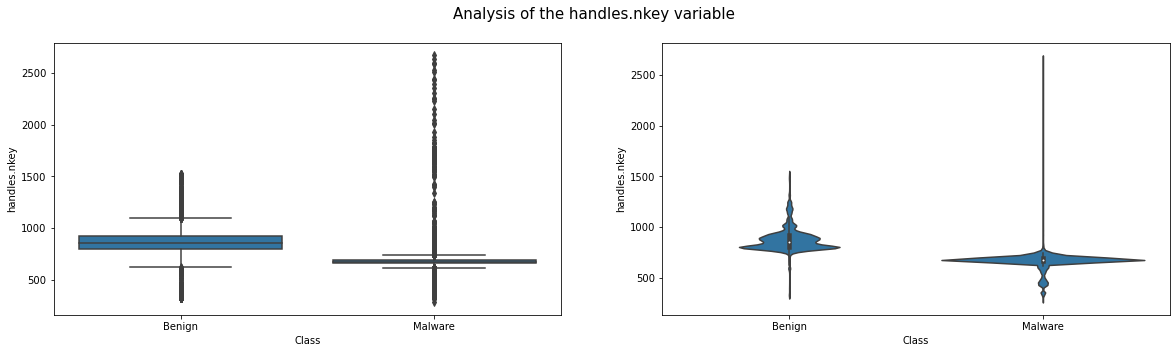
*Figure 22: Analysis of the handles.avg\_handles\_per\_proc* *variable*

Most benign values of the *handles.avg\_handles\_per\_proc* variable are within the range of 200-350. If the value is below 200 or above 350, it could be a malware attack.



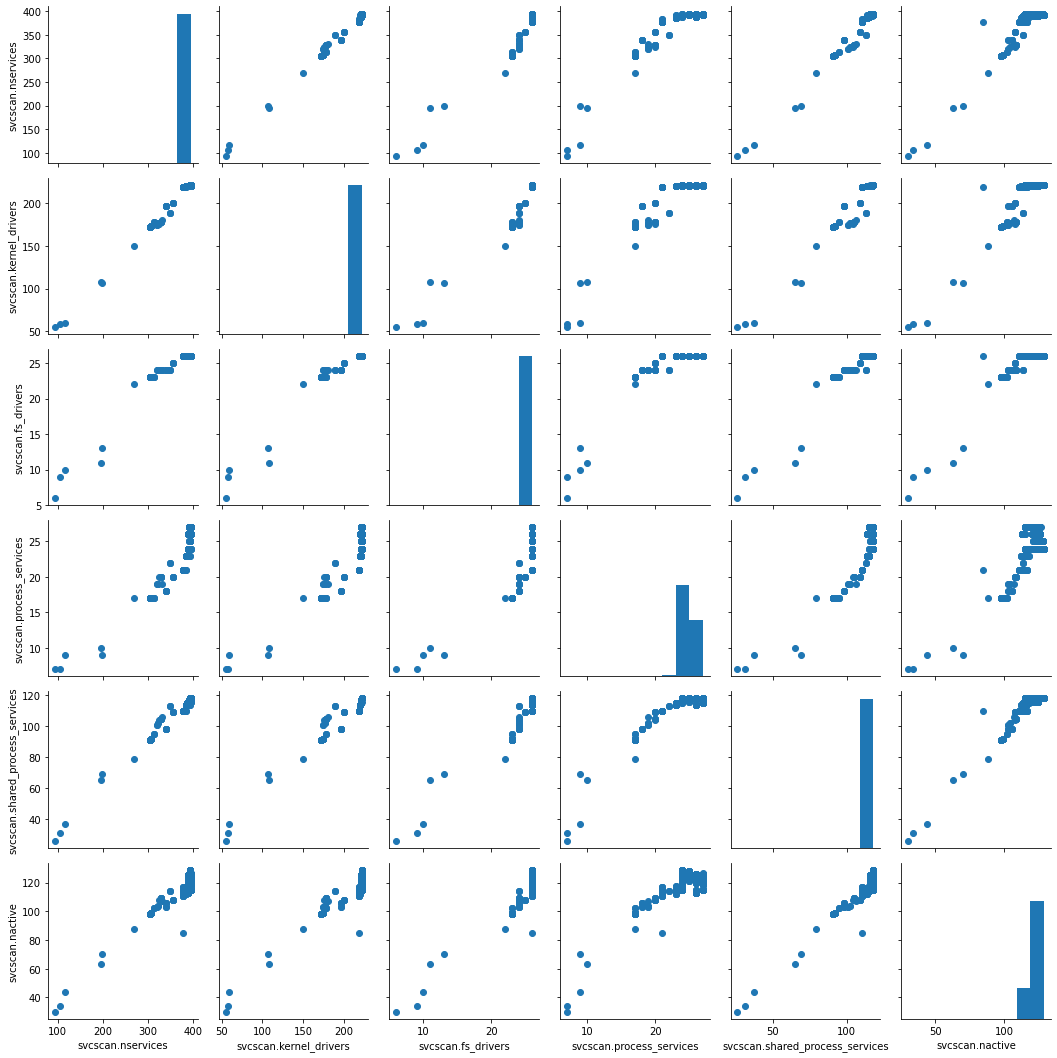
*Figure 23: Analysis of the svcscan.process\_services variable*

The svcscan.process\_services variable of malware attacks have values less than 25. If the value of the svcscan.process\_services variable is below 25, there is a high chance that it is a malware attack. If it is above 25, then it is benign. From the violin plot, most of the benign values of the svcscan.process\_services variable are within the range of 25-27. While most of the malware values of the svcscan.process\_services variable are 24. From the summary statistic, the maximum value of the malware class of the svcscan.process\_services variable is 24, while the minimum is 7. The maximum value of the benign class of the svcscan.process\_services variable is 27, while the minimum is 24.



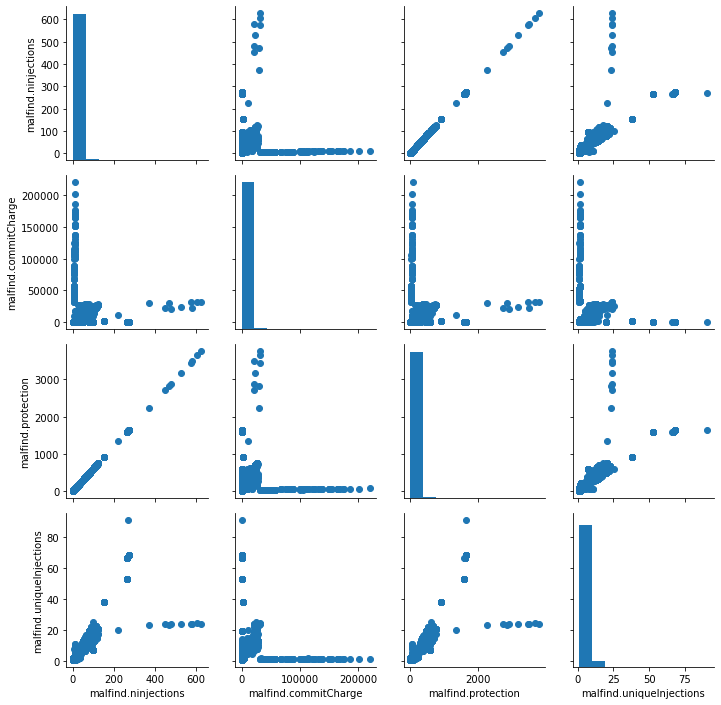
*Figure 24: Analysis of the handles.nkey variable*

Most of the malware values of the handles.nkey variable are within the range of 500 - 800. While most of the benign values of the handles.nkey variable are within the range of 800-1000. Also, the values of the handles.nkey variable that are above 1500 are malware attacks.



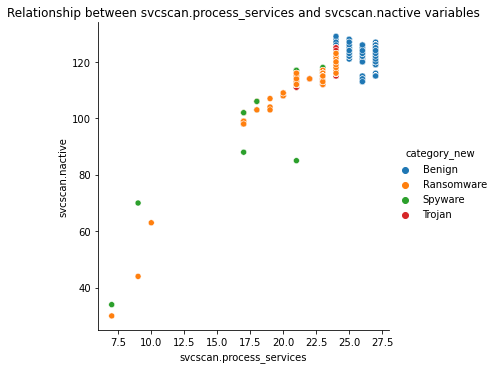
*Figure 25: Plot matrix of some variables*

From the plot matrix above, there is a positive correlation between the following variables: 'svcscan.nservices', and 'svcscan.kernel\_drivers', 'svcscan.fs\_drivers', and 'svcscan.nservices'. 'svcscan.fs\_drivers' and 'svcscan.kernel\_drivers', 'svcscan.shared\_process\_services' and 'svcscan.nservices', 'svcscan.shared\_process\_services' and 'svcscan.kernel\_drivers', 'svcscan.shared\_process\_services' and 'svcscan.nactive', 'svcscan.nactive' and 'svcscan.kernel\_drivers', 'svcscan.nactive' and 'svcscan.nservices', etc. As the value of one variable increases, the value of the other variable increases as well.

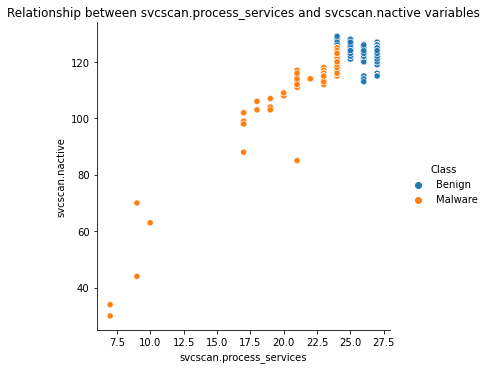


*Figure 26: Plot matrix of some variables*

From the plot matrix above, there is a positive correlation between the 'malfind.protection' and 'malfind.ninjections' variables. From the scatter plot of the 'malfind.ninjections' and 'malfind.uniqueInjections' variables, between 0 and 200 of the 'malfind.ninjections' variable, there is a positive increase in both variables. But after 200, the values of 'malfind.uniqueInjections' stopped increasing as the values of 'malfind.ninjections' increased. Also after the value 40 of the 'malfind.uniqueInjections' variable, as the values of 'malfind.uniqueInjections' increased, the values of 'malfind.ninjections' did not increase. The same thing happened between the 'malfind.protection' and 'malfind.uniqueInjections' variables.

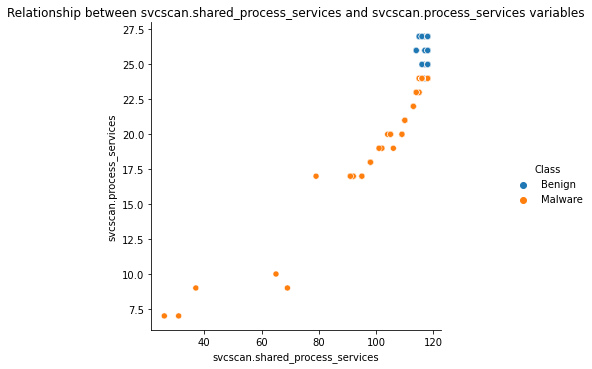


*Figure 27: Scatter plot of the svcscan.process\_services and svcscan.nactive variables against the category\_new variable*



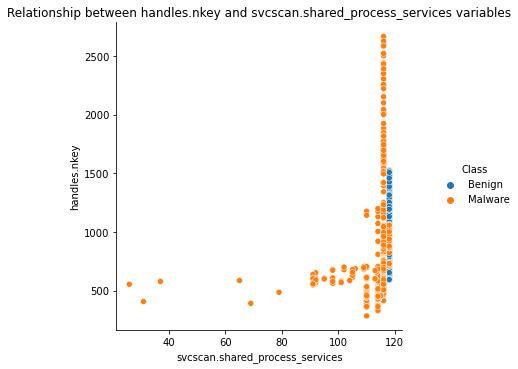
*Figure 28: Scatter plot of the svcscan.process\_services and svcscan.nactive variables against the class variable*

From the scatter plot above, it is easy to identify the different clusters. Most of the benign records have values greater than 22.5 of the svcscan.process\_services and 110 of the svcscan.nactive variable. Most of the ransomware attacks are within 17.5 - 25 value of the svcscan.process\_services variable and 90 - 120 of the svcscan.nactive variable. All the malware attacks have values less than 25.0 (svcscan.process\_services variable) and values less than 120 (svcscan.nactive).



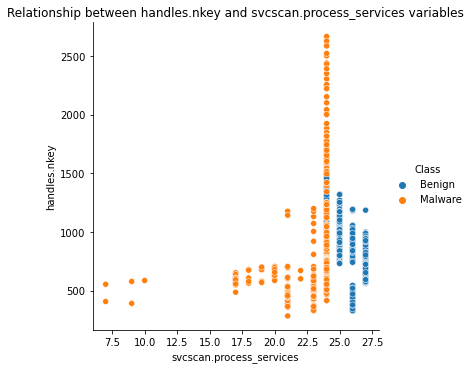
*Figure 29: Scatter plot of the svcscan.process\_services and svcscan.shared\_process\_services variables against the class variable*

From the scatter plot between the svcscan.shared\_process\_services and svcscan.process\_services variables, we see that values below 24 (of the svcscan.process\_services variable) are malware attacks. While values above 24 are benign. Also, the benign records are also above 110 (of svcscan.shared\_process\_services variable).



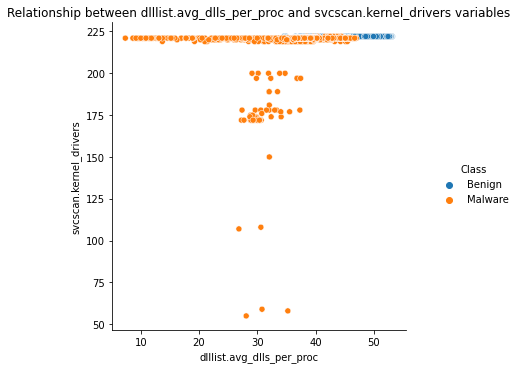
*Figure 30: Scatter plot of the handles.nkey and svcscan.shared\_process\_services variables against the class variable*

The values above 118 of the svcscan.shared\_process\_services variable but below 700 of the handles.nkey variable are benign. Also, the values above 118 of the svcscan.shared\_process\_services variable but above 1200 of the handles.nkey variable are benign. Most of the values below 118 of the svcscan.shared\_process\_services variable are malware attacks.



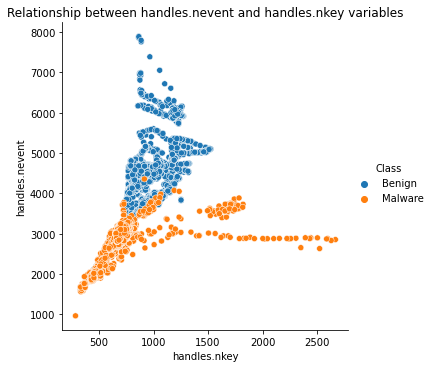
*Figure 31: Scatter plot of the handles.nkey and svcscan.process\_services variables against the class variable*

Most of the benign records are above 24 (of the svcscan.process\_services variable) but below 1500 of the handles.nkey.



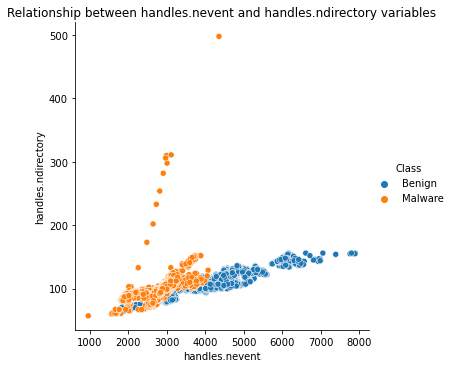
*Figure 32: Scatter plot of the dlllist.avg\_dlls\_per\_proc and svcscan.kernel\_drivers variables against the class variable*

Most of the benign values are above 45 (of the dlllist.avg\_dlls\_per\_proc) and above 200 of the svcscan.kernel\_drivers variable.



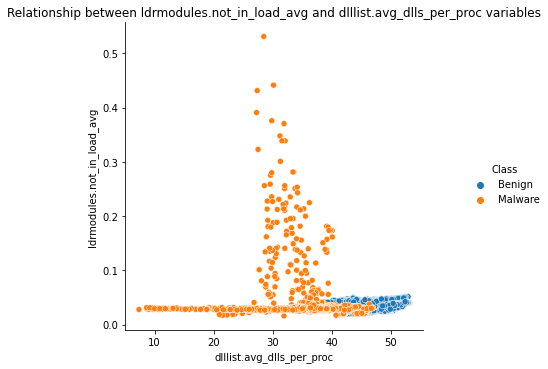
*Figure 33: Scatter plot of the handles.nkey and handles.nevent variables against the class variable*

The benign values are above 3500 of the handles.nevent variable but between the range of 600 - 1600 of handles.nkey variable. Malware attacks are below 4000 of the handles.nevent variable.



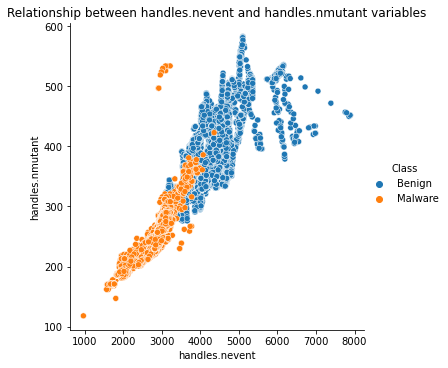
*Figure 34: Scatter plot of the handles.ndirectory and handles.nevent variables against the class variable*

Most of the benign class are below 200 of the handles.ndirectory variable, but above 4000 of the handles.nevent variable. All the values above 200 of the handles.ndirectory variable belong to the malware class.



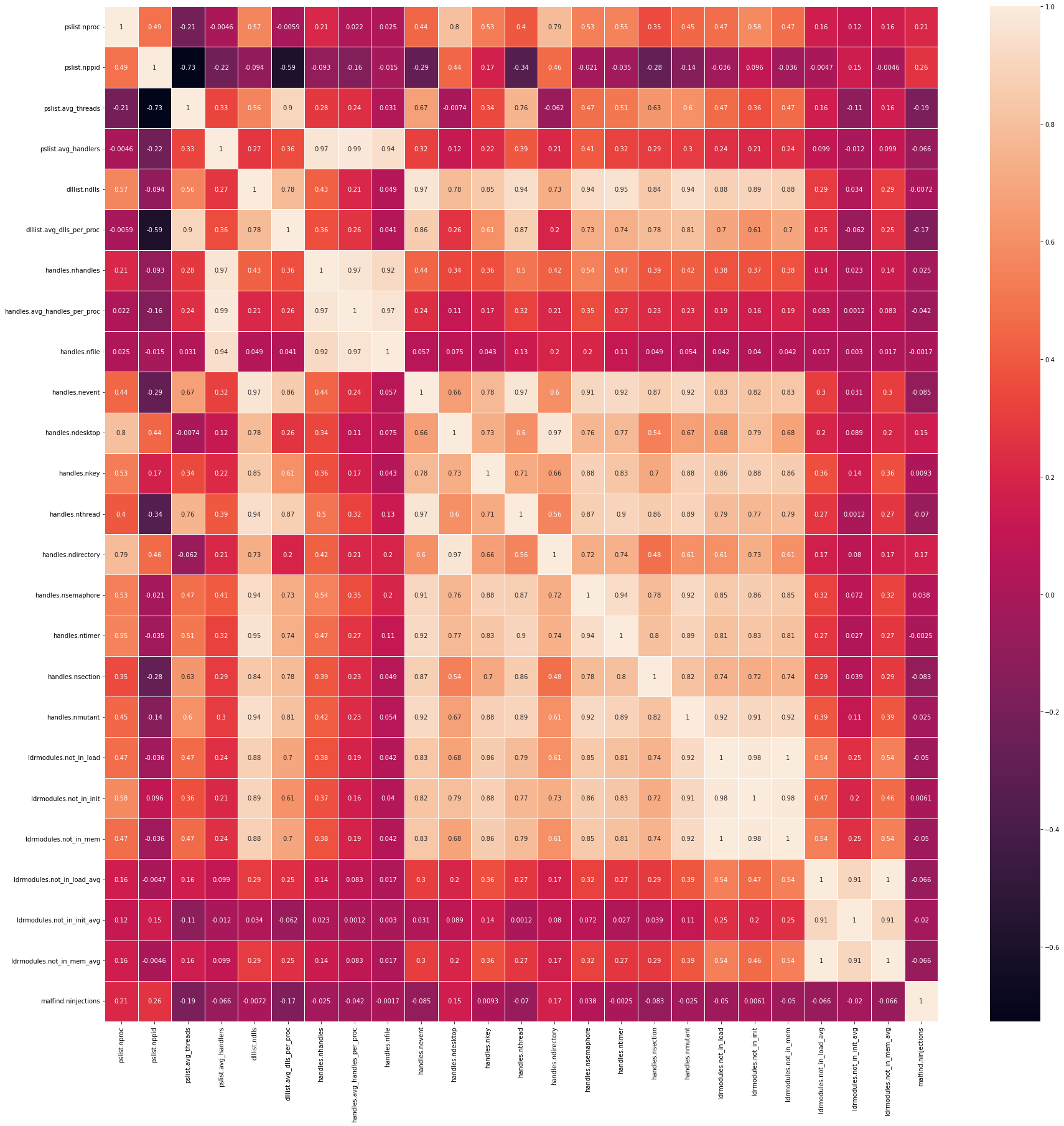
*Figure 35: Scatter plot of the ldrmodules.not\_in\_load\_avg and dlllist.avg\_dlls\_per\_proc variables against the class variable*

Most of the records in the benign class have values above 40 for the dlllist.avg\_dlls\_per\_proc but below 0.1 (ldrmodules.not\_in\_load\_avg). All values above 0.1 of the ldrmodules.not\_in\_load\_avg variable are malware attacks. Also, values below 30 of the dlllist.avg\_dlls\_per\_proc variable are malware attacks.



*Figure 36: Scatter plot of the handles.nmutant and handles.nevent variables against the class variable*

The benign class have values greater than 3000 (of the handles.nevent) and greater than 300 of the handles.nmutant variable. All the records that have values greater than 4500 (of the handles.nevent) are benign. All values that are less than 3000 of the (of the handles.nevent) are malware. There is a small cluster of malware class between 500 - 600 of the handles.nmutant variable and 2500 - 3500 of the handles.nevent variable.



*Figure 37: Confusion matrix heat map for some variables*

There is a high positive correlation between most of the variables. For instance, the correlation coefficient between pslist.avg\_threads and dlllist.avg\_dlls\_per\_proc is 0.9. The correlation coefficient between pslist.avg\_handlers and handles.avg\_handles\_per\_proc is 0.99. The correlation coefficient between handles.nhandles and handles.nfile is 0.92.

**PREDICTIVE ANALYSIS**

In this section, I will train a linear classifier and a non-linear classifier to detect malicious and benign attacks. I removed the *'category\_new',* and *'type'* variables. I won’t need them to train a machine learning model. After removing these features, the dataset had 58062 records and 52 variables.

I label encoded the *Class* variable. Before training a machine learning model, the text values must be converted to numbers. The benign value is encoded as 0, while the malware value is encoded as 1.

**EXPERIMENT 1 – Train Model with all the Features**

I separated the features and the target variable. The *Class* variable is the target variable. The features are *'pslist.nproc', 'pslist.nppid', 'pslist.avg\_threads', 'pslist.avg\_handlers', 'dlllist.ndlls', 'dlllist.avg\_dlls\_per\_proc', 'handles.nhandles', 'handles.avg\_handles\_per\_proc', 'handles.nfile', 'handles.nevent', 'handles.ndesktop', 'handles.nkey', 'handles.nthread', 'handles.ndirectory', 'handles.nsemaphore', 'handles.ntimer', 'handles.nsection', 'handles.nmutant', 'ldrmodules.not\_in\_load', 'ldrmodules.not\_in\_init', 'ldrmodules.not\_in\_mem', 'ldrmodules.not\_in\_load\_avg', 'ldrmodules.not\_in\_init\_avg', 'ldrmodules.not\_in\_mem\_avg', 'malfind.ninjections', 'malfind.commitCharge', 'malfind.protection', 'malfind.uniqueInjections', 'psxview.not\_in\_pslist', 'psxview.not\_in\_eprocess\_pool', 'psxview.not\_in\_ethread\_pool', 'psxview.not\_in\_pspcid\_list', 'psxview.not\_in\_csrss\_handles', 'psxview.not\_in\_session', 'psxview.not\_in\_deskthrd', 'psxview.not\_in\_pslist\_false\_avg', 'psxview.not\_in\_eprocess\_pool\_false\_avg', 'psxview.not\_in\_ethread\_pool\_false\_avg', 'psxview.not\_in\_pspcid\_list\_false\_avg', 'psxview.not\_in\_csrss\_handles\_false\_avg', 'psxview.not\_in\_session\_false\_avg', 'psxview.not\_in\_deskthrd\_false\_avg', 'modules.nmodules', 'svcscan.nservices', 'svcscan.kernel\_drivers', 'svcscan.fs\_drivers', 'svcscan.process\_services', 'svcscan.shared\_process\_services', 'svcscan.nactive', 'callbacks.ncallbacks', 'callbacks.nanonymous',* and *'callbacks.ngeneric'.*

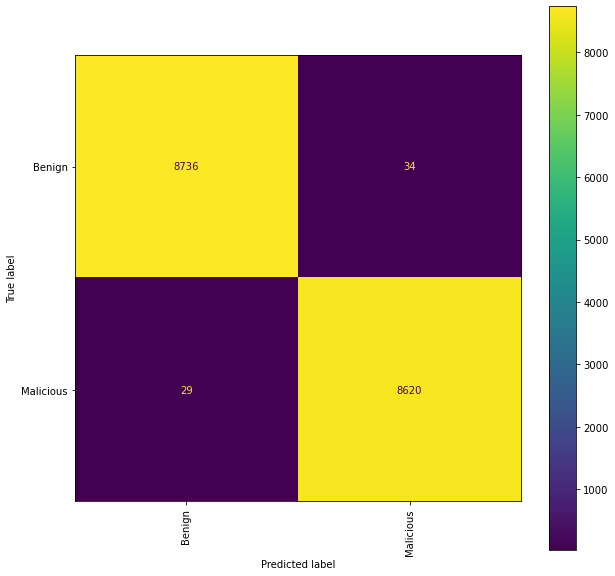
I splitted the dataset into the training set and testing set. I used 70% of the data for training and 30% to evaluate the model’s performance. There are 40643 records in the training set. There are 17419 records in the testing set. Since the variables of the dataset are not on the same scale, I applied normalization to the data set using the *MinMaxScaler()* class. Each variable is scaled to have a maximum value of 1 and a minimum value of 0. Finally, I trained a logistic regression model and a random forest model on the data.

**Results**

The result of the models on the test set is shown in the table below.

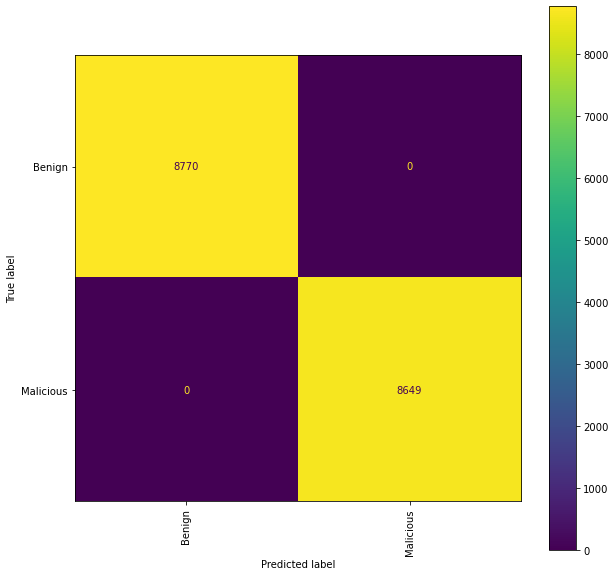
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy (%) | Precision (%) | Recall (%) | F1-score (%) |
| Logistic Regression | 99.64 | 99.64 | 99.64 | 99.64 |
| Random Forest | 100 | 100 | 100 | 100 |

*Table 1: Metrics of models on the test set*



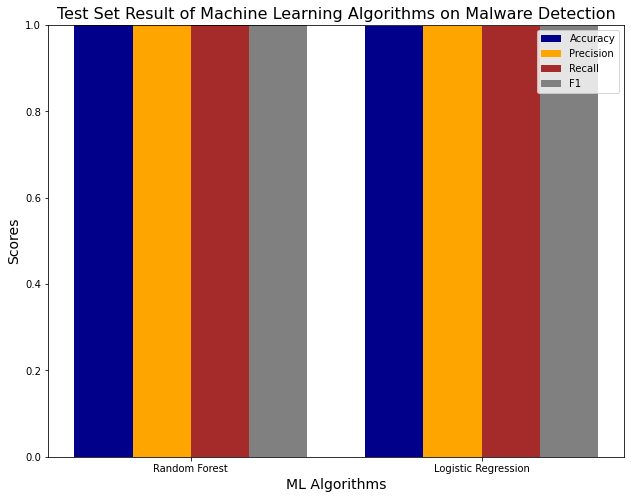
*Figure 38: Confusion matrix of the Logistic Regression model*

From the confusion matrix above, the number of true positives is 8620, the number of false positive is 34, the number of true negatives is 8736 and the number of false negatives is 29. In other words, the model correctly classified 8620 cases as malware. It wrongly classified 34 benign attacks as malware attacks. It correctly classified 8736 cases as benign. It wrongly classified 29 malware attacks as benign.



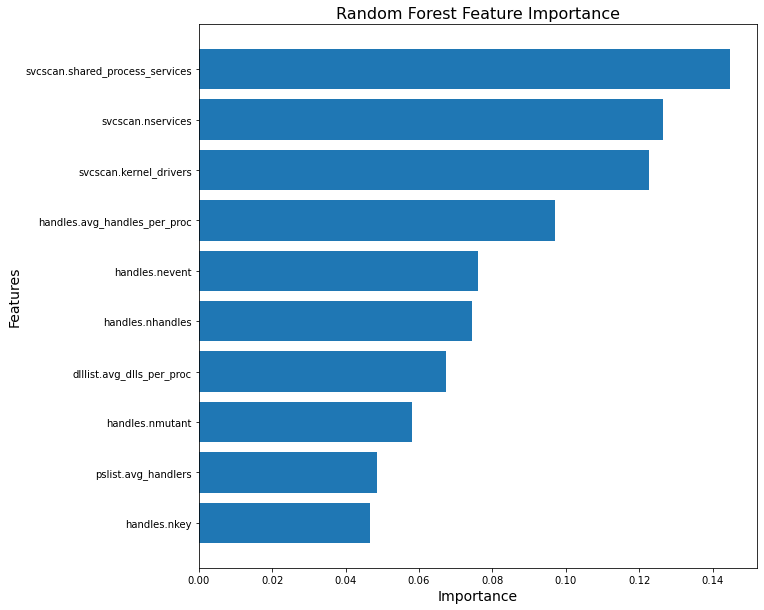
*Figure 39: Confusion matrix of the Random Forest model*

From the confusion matrix above, the number of true positives is 8649, the number of false positive is 0, the number of true negatives is 8770 and the number of false negatives is 0. Initially, I assumed that the random forest model was overfitting. So I performed hyperparameter tuning using GridSearchCV. The parameters of the fine-tuned random forest model are 'criterion': 'gini' and 'n\_estimators': 100. After fine-tuning the model, I got the same result. The random forest model correctly classified all the malware and benign attacks.



*Figure 40: Clustered bar chart comparing the random forest model and the logistic regression model*

From the results above, the random forest model performed better than the logistic regression model.



*Figure 41: A plot of the top 10 features*

The top 5 features *are 'svcscan.shared\_process\_services', 'svcscan.nservices', 'svcscan.kernel\_drivers', 'handles.avg\_handles\_per\_proc',* and *'handles.nevent'.*

**EXPERIMENT 2 – Train Model Without Correlated Features**

There are 29 features whose correlation coefficient is greater than 0.9. These features are *'handles.nhandles', 'psxview.not\_in\_ethread\_pool\_false\_avg', 'ldrmodules.not\_in\_init', 'malfind.uniqueInjections', 'handles.ntimer', 'dlllist.avg\_dlls\_per\_proc', 'psxview.not\_in\_pspcid\_list\_false\_avg', 'svcscan.shared\_process\_services', 'ldrmodules.not\_in\_init\_avg', 'handles.avg\_handles\_per\_proc', 'handles.ndirectory', 'handles.nevent', 'ldrmodules.not\_in\_mem\_avg', 'handles.nthread', 'psxview.not\_in\_csrss\_handles\_false\_avg', 'psxview.not\_in\_deskthrd', 'psxview.not\_in\_session', 'handles.nmutant', 'ldrmodules.not\_in\_mem', 'psxview.not\_in\_session\_false\_avg', 'psxview.not\_in\_pspcid\_list', 'handles.nfile', 'svcscan.fs\_drivers', 'ldrmodules.not\_in\_load', 'psxview.not\_in\_deskthrd\_false\_avg', 'psxview.not\_in\_csrss\_handles', 'malfind.protection', 'psxview.not\_in\_pslist\_false\_avg',* and *'handles.nsemaphore'.*

Initially, the dataset had 58062 records and 52 features. But after removing the 29 features with high correlation coefficient, there were 58062 records and 23 features. The features are *'pslist.nproc', 'pslist.nppid', 'pslist.avg\_threads', 'pslist.avg\_handlers', 'dlllist.ndlls', 'handles.ndesktop', 'handles.nkey', 'handles.nsection', 'ldrmodules.not\_in\_load\_avg', 'malfind.ninjections', 'malfind.commitCharge', 'psxview.not\_in\_pslist', 'psxview.not\_in\_eprocess\_pool', 'psxview.not\_in\_ethread\_pool', 'psxview.not\_in\_eprocess\_pool\_false\_avg', 'modules.nmodules', ‘svcscan.nservices', 'svcscan.kernel\_drivers', 'svcscan.process\_services', 'svcscan.nactive', 'callbacks.ncallbacks', 'callbacks.nanonymous'*, and *'callbacks.ngeneric'.*

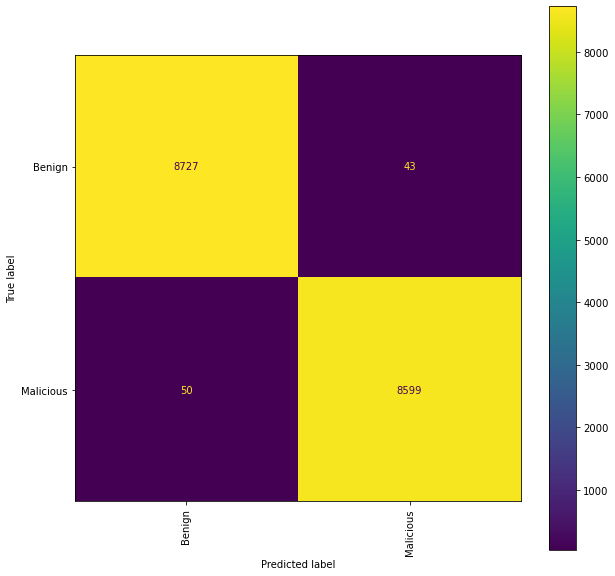
I splitted the dataset into the training set and testing set. I used 70% of the data for training and 30% to evaluate the model’s performance. There are 40643 records in the training set. There are 17419 records in the testing set. I applied normalization to the data set using the *MinMaxScaler()* class. Each variable is scaled to have a maximum value of 1 and a minimum value of 0. I trained a logistic regression model and a random forest model on the data without highly correlated features.

**Results**

The result of the models on the test set is shown in the table below.

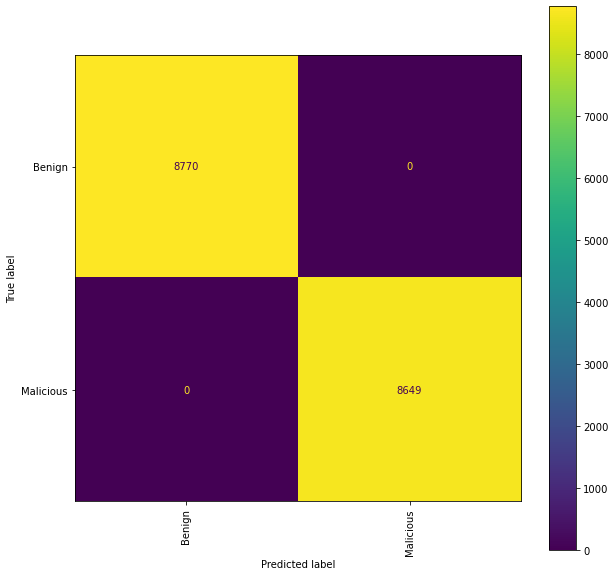
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy (%) | Precision (%) | Recall (%) | F1-score (%) |
| Logistic Regression | 99.47 | 99.47 | 99.47 | 99.47 |
| Random Forest | 100 | 100 | 100 | 100 |

*Table 2: Metrics of models on the test set*



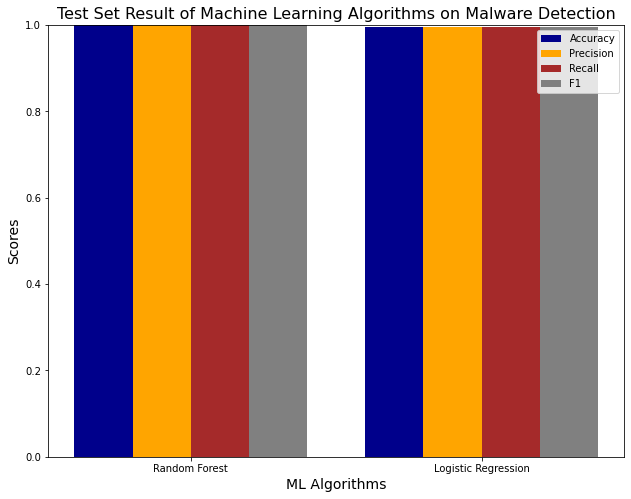
*Figure 42: Confusion matrix of the Logistic Regression model*

From the confusion matrix above, the number of true positives is 8599, the number of false positive is 43, the number of true negatives is 8727 and the number of false negatives is 50. In other words, the model correctly classified 8599 cases as malware. It wrongly classified 43 benign attacks as malware attacks. It correctly classified 8727 cases as benign. It wrongly classified 50 malware attacks as benign.



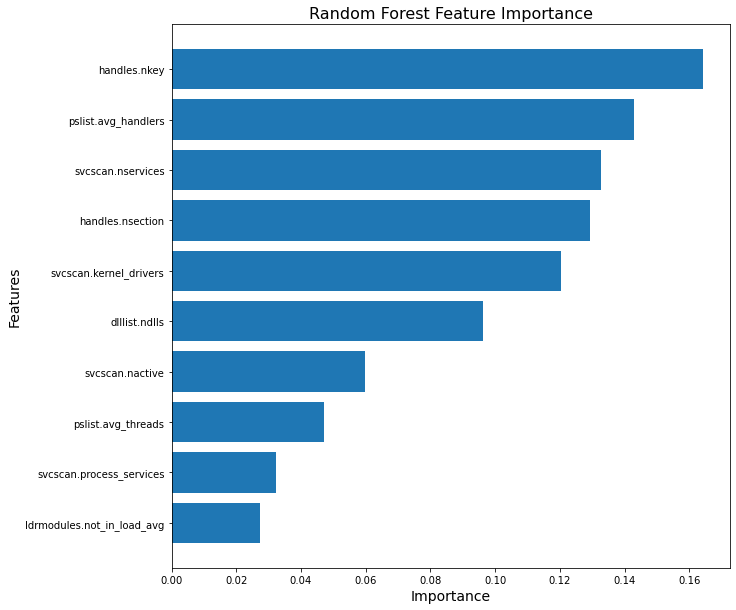
*Figure 43: Confusion matrix of the Random Forest model*

From the confusion matrix above, the number of true positives is 8649, the number of false positive is 0, the number of true negatives is 8770 and the number of false negatives is 0. I performed hyperparameter tuning using GridSearchCV. The parameters of the fine-tuned random forest model are 'criterion': 'gini' and 'n\_estimators': 100. After fine-tuning the model, I got the same result. The random forest model correctly classified all the malware and benign attacks



*Figure 44: Clustered bar chart comparing the random forest model and the logistic regression model*

From the results above, the random forest model performed better than the logistic regression model.



*Figure 45: A plot of the top 10 features*

The top 5 features *are 'handles.nkey', 'pslist.avg\_handlers', 'svcscan.nservices', 'handles.nsection',* and *'svcscan.kernel\_drivers'.*

**RECOMMENDATION**

1. If the value of the *svcscan.shared\_process\_services* variable is below 100, it could be a malware attack.
2. If the value of the *svcscan.nservices* variable is less than 350, there is a high chance that it is a malware attack.
3. If the value of the *svcscan.kernel\_drivers* variable is less than 200, there is a high chance that it is a malware attack.
4. Most benign values of the *handles.avg\_handles\_per\_proc* variable are within the range of 200-350. If the value is below 200 or above 350, it could be a malware attack.
5. Most benign records of the *handles.nevent* variable are within the range of 3500-5000. If the value is below 3000, it could be malicious attacks.

In addition, if the value of *dlllist.avg\_dlls\_per\_proc* is below 40, it could be a malware attack. If the value of *dlllist.ndlls* is below 1500, there is a high chance that it is a malicious attacks. These variables should be monitored. Any reading outside it's normal range is an anomaly and it could indicate a malicious attack.

**CONCLUSION**

From the results above, it is clear that memory-based technique is very useful in detecting obfuscated malware like ransomware, spyware and Trojan. In other words, we can use machine learning models to detect obfuscated malware by using memory data as features.

Even with few features, machine learning models are good at finding anomalies in the memory data. The results of the random forest model (non-linear classifier) was not affected when the correlated features were removed. But the result of the logistic regression model (linear classifier) was affected when the correlated features were removed. The performance of the logistic regression model dropped a bit when the highly correlated features were removed. The number of false positives of the logistic regression model increased from 34 to 43. While the number of false negatives increased from 29 to 50.

If you want to deploy this model, I would recommend using the random forest model that was trained on fewer features. It will be easier to manage in the production environment since we are working with 23 features instead of 52 features.

**REFERENCES**

* AWS Documentation (n.d.). *Amazon Simple Storage**Service.* Viewed 31 July 2022. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/creating-bucket.html>
* Kochovski, A. (2022, May). *The Risks and Benefits of Cloud Storage in 2022.* Cloudwards. Viewed 30 July 2022. <https://www.cloudwards.net/the-risks-and-benefits-of-cloud-storage/>
* Thompson, J. (2022, January). *Advantages and Disadvantages of Cloud Storage in 2022.* V2 Cloud. Viewed 30 July 2022. <https://v2cloud.com/blog/advantages-disadvantages-cloud-storage>.
* Tristan Carrier, Princy Victor, Ali Tekeoglu, Arash Habibi Lashkari,” Detecting Obfuscated Malware using Memory Feature Engineering”, The 8th International Conference on Information Systems Security and Privacy (ICISSP), 2022
* University of New Brunswick (n.d.). *About the Canadian Institute for Cybersecurity*. Viewed 31 July 2022. <https://www.unb.ca/cic/about/index.html>.
* Usman M. (2018, October 31). *Applying Filter Method in Python for Feature Selection*. Viewed 29 July 2022. <https://stackabuse.com/applying-filter-methods-in-python-for-feature-selection/>