Supervised Learning Algorithm used for LOLBins detection in Linux Machines

Ângello Cássio Vasconcelos Oliveira
University of Brasília
Electrical Engineering Department
Brasília (DF), Brasil

Daniel Chaves Café
University of Brasília
Electrical Engineering Department
Brasília (DF), Brasil

Abstract — **Living Off the Land Binaries (LOLBins) is a cyber intrusion technique in which the attacker exploits legitimate operating system binaries to carry out criminal actions. This work proposes to compare different machine learning techniques to automatically detect LOLBins attacks. The command lines were preprocessed using Natural Language Processing (NLP) algorithms. The effectiveness of the methods was evaluated taking into consideration data balancing techniques such as Random Oversampling/Undersampling and SMOTE. Performance metrics such as accuracy and F1-Score were used to compare the different techniques. In conclusion, it was verified that the Decision Tree (DT) and Random Forest (RF) algorithms were superior to the others. Furthermore, using a command window as input generated better results than evaluating isolated command lines.**

Keywords – linux; machine learning, intrusion detection, Living Off the Land, LOTL, LOLBins.

I. INTRODUCTION

In 2023, a relevant vulnerability was identified and exploited in *Barracuda Email Security Gateway* (ESG) [12] [13]. This security failure consists in sending a malicious *.tar* file as an email attachment. When processed by the ESG system, the *.tar* file exploited a flaw in the validation and sanitization of file names within the system.

This security breach opened a vector to Remote Code Execution (RCE), which was abused using Living Off the Land Binaries (LOLBins) techniques. Analyzing the Indicators of Compromise (IOC) it was observed that the attackers used the code 1 to execute commands in the target through a *reverse shell*. With this purpose, the attackers used the *openssl* command to establish this reverse connection and *mkfifo* to create a named pipe to redirect the *sh* commands and its results between the target and the attacker's Command and Control (C2).

```
setsid sh -c "mkfifo /tmp/p; sh -i </tmp/p 2>&1 | openssl s_client -quiet -connect 107.148.149[.]156:8080 >/tmp/p 2>/dev/null; rm /tmp/p"
```

Code 1. Bash Script used in Barracuda exploit.

LOLBins [16] is a technique in which legitimate operating system binaries can be exploited by malicious individuals to subvert misconfigured systems. This technique, as elucidated by Ding et al. [7], is a furtive method used by attackers to harder their detection and has been increasingly seen as part of the APT's (Advanced Persistent Threats) Tactics, Techniques and Procedures (TTP) [1].

The central goal of this study is to carry out a binary classification of Linux commands, with the specific aim of identifying tactics associated with the use of LOLBins. To achieve this, combinations of several Machine Learning models will be employed, including Random Forest, Neural Networks and Decision Trees, concomitantly with the use of Natural Language Processing (NLP) techniques, such as Bag of Words, TF-IDF and Doc2Vec. Furthermore, to overcome the challenge of data imbalance, artificial database balancing algorithms will be implemented.

Thus, the analysis focused on commands originating from the Linux operating system, with a significant portion of the data originating from records in the file "*.bash_history*", the same way portion used by Ngan [14] in his work, and the list of commands from the GTFOBins website [16]. This restricted the information available to command text only. Given this contextual limitation, the solution proposed by Ngan [14] was used, which involved analyzing a window of *N* previous commands to provide more context to the lack of detailed data about each command.

As highlighted by Kotsiantis et al. [9], the challenge of unbalanced data is increasingly common, especially in practical situations that involve the detection of rare but relevant events. This scenario is similar to the context of identifying malicious commands or malwares. Due to the limited and relatively small list of LOLBins, it was necessary to employ artificial dataset balancing techniques given that these commands represented only 1,601% of the total. To mitigate this disparity, several techniques were explored, including *Random Undersampling (RUS)* , *Random Oversampling (ROS)* and *Near Miss*, aiming to balance data distribution and improve the effectiveness of detection models. However, for a better analysis of LOLBins, it was necessary to consider not only Machine Learning models and Artificial Balancing algorithms, but also the Natural Language Processing (NLP) algorithms used to represent the command's strings. This interaction between different approaches directly influenced the final metrics of each test.

Data preprocessing was carried out in a similar way to Ngan's work. However, improvements were made to some functions to avoid possible loss of information that could occur when dealing with environment variables and file paths.

In this context, an analysis of relevant studies in machine learning applied to the detection of cyber threats will be carried out, as well as the methodology adopted for the development and automation of codes, which facilitate the training process of machine learning models. Finally, the results achieved through these approaches will be presented and discussed providing insights and contributions to the field of cybersecurity.

II. Related Work

The vast majority of machine learning work in the cybersecurity field is focused on the area of malware detection. For example, Chumachenko (2017) [6] explored several Machine Learning algorithms for the purpose of detecting and classifying malware. His research concluded that, with his dataset, the most accurate algorithms were Random Forest, followed by J48. Furthermore, Rathore (2018) [17] compared the results between the use of Machine Learning and Deep Learning in malware detection, concluding that Random Forest outperformed Deep Neural Networks. In both studies, Random Forest proved to be a promising option for threat detection. As Rathore, Ding [7] implemented Deep Learning techniques, this time, to detect LOLBins, reaching an accuracy of 99.45%.

Work aimed at detecting LOLBins in general, in which was included other systems such as Windows, for example, addresses this interaction between different ways of representing texts originating from command lines. An example of this was Ogun [15] who implemented a way of assigning features that he called cmd2vec. In his work, Ongun [15] submited the tokens for each command to the Doc2Vec or FastText algorithm for vectorization. After that, he trained his first database using the Random Forest algorithm and calculates the average probability that each token appears on a leaf of the tree with a final label representing the LOLBins commands. As Windows LOLBins commands mostly have a greater number of tokens than regular commands or even the Linux LOLBins, Ongun created a new database where, for each entry, he used the metrics obtained through previous training, creating a new entry containing 3 tokens with the highest probability, 3 with the lowest probability, 3 with the medium grouping and the 3 with the highest probability scores. Finally, he also included a count of the entry tokens and the rarest ones, as he noticed that in his database the binaries used in attacks sometimes appeared only a few times.

Boros [3] also tried to improve the representation of the command line strings. He assigned key characteristics that represented certain actions in each input to better interpret the commands.

Although Thuy Ngan's master's thesis [14] (DAO, 2020) focused on the classification of Unix commands in general, categorizing them by risk levels without focusing specifically on LOLBins, the present work appropriates some of the techniques addressed by Ngan to investigate LOLBins attacks further, such as using command windows to represent the context of commands.

To achieve his goal, Ngan [14] collected 660 *.bash_history* files from GitHub and used logs from two honeypots from another study carried out at his university. The total corpus was 905,405 commands. The availability of its database represented another significant contribution of his work. Accessing this data set enabled a direct and effective comparison with the performance of the algorithms used in the present study.

In a first analysis, Ngan performed a binary classification of the data, assuming that GitHub commands were legitimate, while those coming from honeypots were considered malicious. In a second stage, all data was subjected to a labeling function, categorizing each command into risk indices.

Their results, as shown in Tables I and II, demonstrate that the use of a context, in this case a window of 3 commands, combined with changing the NLP algorithm improved accuracy in almost 10%. In all scenarios, both in the binary classification approach and in the risk classification, the NLP algorithm that had the best performance was Doc2Vec.

TABLE I. Ngan's results in Binary Classification

Context	***Representation***	***Classifier***	***Accuracy***	***False Neg.***
1-Command	1-gram + BoW	KNN	89.13%	28,878
3-Command	1-gram + TF-IDF	Linear SVC	98.13%	2,292
	3-gram + TF-IDF		98.27%	2,066
	3-gram + Doc2Vec		96.71%	1,499

TABLE II. Ngan's results in Risk Level Classification

Representation	***Classifier***	***Accuracy***
Count-Vector	Logistic Regression	95,05%
Doc2Vec		99,58%

At the end of each training, the predictions were submitted to a confusion matrix, where the False Negative and False Positive rates were measured. False Negatives are particularly more harmful as the incorrect classification of a malicious command can have significant impacts on a given system.

Therefore, this work seeks to apply reasoning similar to Ngan's but with a focus only on the detection of LOLBins in Linux, proposing some improvements over his command processing approach and proving some insights obtained in previous works, mainly regarding the need not to analyze commands in isolation.

III. Methodology

To analyze the LOLBins, the benign portion of the Ngan database was used together with the list of commands obtained from the website *https://gtfobins.github.io* [16]. This GitHub repository offers a JSON file containing the commands listed on the site, accessible via the "*code*" key.

The preprocessing approach followed a similar pattern to Ngan. However, Ngan's implementation had some vulnerabilities. As for the representation of the complete paths of a command, they were always suppressed, having, for example, "*/bin/bash -p*" being transformed into "*_PATH_ -p*", which could result in the loss of important information. Furthermore, a frequent use of environment variables was observed in LOLBins list. Given that variables can be named in

different ways, this could lead the models to consider them different while they represent the same action.

```
def replace_file_path(text):
    # Modified to replace only the directory, not the file at the end
    return re.sub(r"((?<= )[^ ]*/)([^/ ]*)", r"_PATH_/\2", text)

def replace_env_variables(text):
    # Replace $String and $_ENV for String= ou _ENV_=.
    text = re.sub(r"\$([A-Za-z_][A-Za-z0-9_]*)", r"$_ENV", text)
    text = re.sub(r"([A-Za-z_][A-Za-z0-9_]*)=", r"_ENV_=", text)
    return text
```

Code 2. Python script for text manipulation

To overcome these information losses, the functions *replace_file_path* and *replace_env_variables* were changed according to Code 2 to respectively replace only the directory, not the full path with PATH string and replace variable declarations with the ENV string.

A. *Unbalanced Learning*

After processing the data, it was observed that the database was excessively unbalanced with the part of the data that represented GTFOBins being only 1,601 % of the total. So, if any model just guessed a value, it would be correct approximately 98% of the time. For this, some algorithms were used to artificially balance databases such as Random Oversampling and Random Undersampling.

Random Oversampling (ROS) randomly upsamples the data that is outnumbered while Random Undersampling (RUS) does the opposite, reducing the majority sampling. As reported by Weiss [19], these techniques have their shortcomings such as Undersampling leading to loss of information due to discarding data and Oversampling which has the risk of leading to overfitting.

To improve comparative sampling, other algorithms were applied as well, such as the Synthetic Minority Over-Sampling Technique (SMOTE), which, according to Alamsyah et al. [2], generates new data based on the minority class to balance the dataset. Near Miss, in turn, removes samples from the majority class based on proximity to the minority data, seeking to preserve as much information as possible. One Side Selection, according to Kubat [10], eliminates points distant from the minority class, aiming to improve the representation of this class.

In this way, 02 functions were created in Python that would automate the process of creating balanced data, dividing the data into testing and training, and finally training the data with the chosen model. The *create_dataset_balanceado* function performs data balancing using *random_state = 32* as its only parameter.

Chicco and Jurman (2020, 2021) highlight a limitation in evaluating machine learning models in binary classification tasks with imbalanced datasets. This is because, as mentioned before, in a scenario where data is disproportionately distributed across classes, a model that simply predicts the majority class in all instances would achieve a high accuracy rate despite not having learned to effectively distinguish between the classes. On the other hand, *Matthews Correlation Coefficient* (MCC) and F1-Score are more robust, with F1 being one of the most used metrics in the context of Machine Learning, not only in binary classification but also in multiclass classification [4] [5].

To provide a balanced and clear comparison of the models, a metric called "grade" was created, which is represented by $\frac{Accuracy+2*F1+MCC}{4}$, according to Code 3. In this new metric, the F1 score had a greater weight than the others because, since it considers both accuracy and recall, it is considered useful in scenarios with an unbalanced dataset.

This metric is calculated through the *fit_predict* function, which is responsible for dividing the data into test and training, carrying out the training and finally extracting the Accuracy, F1-Score, MCC and grade metrics. For the purpose of replicating the results, the parameters *test_size=0.3*, *random_state=42* were used to separate the training and testing datasets.

```
def fit_predict(model, X, Y):
    train_cmds, test_cmds, train_labels, test_labels
        = train_test_split( X, Y, test_size=0.3, random_state=42)

    # Training the dataset
    model.fit(train_cmds, train_labels)

    # Predicting the results
    y_pred = model.predict(test_cmds)

    # Gathering the metrics
    acc = accuracy_score(test_labels, y_pred)*100
    f1 = f1_score(test_labels, y_pred)*100
    mcc = 3atthews_corrcoef(test_labels, y_pred)*100

    # Final grade proposed for this article
    nota = (acc + 2*f1 + mcc)/4

    return {"Accuracy": acc, "F1-Score": f1, "MCC": mcc, "Nota": nota}
```

Code 3. Python Script for training and validating a model

B. *Test's Order*

Commands executed in the Linux terminal are saved in the *.bash_history* file. This file stores solely and exclusively the texts of the commands and their ordering. This way, any other information relevant to the context of an attack, such as date and time, user and permissions, is not stored. To overcome this limitation, Ngan (2020) proposed the use of a 'command window' as a method to analyze the operational context of commands. This approach, by considering a sequence of commands instead of evaluating each one in isolation, demonstrated a significant improvement in his results, increasing the accuracy in identifying malicious activity by around 10%.

Based on this understanding, the first analysis scenario was conducted without incorporating contextual analysis, treating each command individually and applying the *Bag of Words*

technique for data modeling. The first reference to this term in the linguistic context was attributed to Zellig Harris [8]. The bag of words (BoW) is a textual representation in vector form, where a set of words is organized into a list, each item corresponding to a specific word. This method counts the occurrences of the words in this list for each text entry, as detailed by Marcelo Silveira in his book on machine learning [18].

The second scenario was conducted using the TF-IDF technique, applied to a sequence of three consecutive commands, thus aligning with the contextual analysis methodology introduced by Ngan. The TF-IDF method is an approach to representing the importance of words in a text. Described by Ding et al. [7], this method uses Term Frequency (TF) combined with Inverse Document Frequency (IDF), where IDF is calculated as $IDF(term) = log\left(\frac{N}{\text{df(term)}}\right)$, seeking to measure the relevance of a term within a document.

Finally, Doc2Vec was used with the same window of 3 commands. Doc2Vec was created by Le and Mikolov (2014) as a simple extension to Word2Vec. As elucidated by Lau and Baldwin [11], the intent of this extension was to create vector representations for documents rather than individual words.

IV. Results

For each scenario, 5 training sessions were carried out, one for each balancing algorithm, for each of the 6 supervised learning models. As seen in Figure 1, grades with the Doc2Vec implementation had the best results.

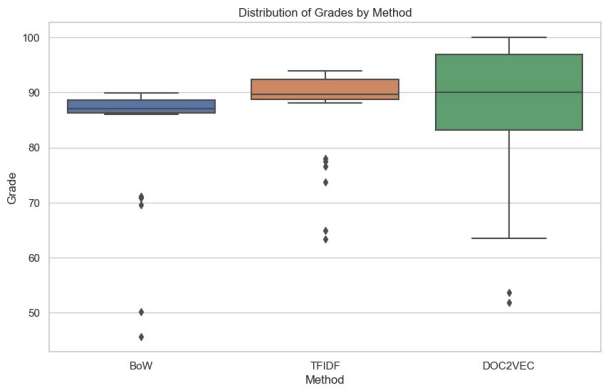

Figure 1. Distribution of grades by method.

A. Oversampling

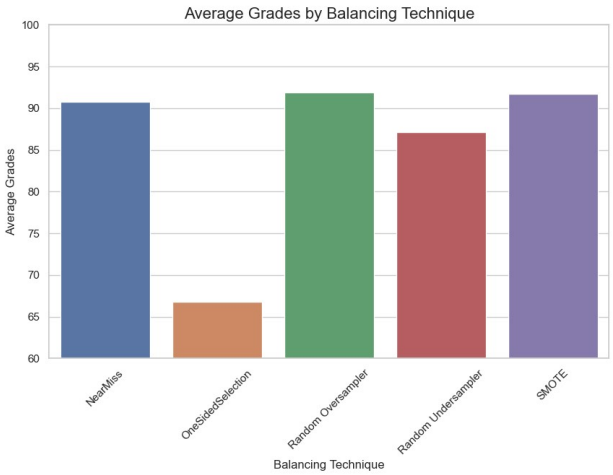

Figure 2. Average grades by balancing techniques.

In the experiments carried out, it was observed that algorithms focused on increasing the representation of the minority class, specifically Random Oversampling and SMOTE, presented superior performance compared to algorithms that aim to reduce the majority class. The graphical analysis, illustrated in Figure 2, reveals the average 'grades' achieved by each balancing method. These data show that the two algorithms mentioned not only sustained high averages but also presented results that were very similar to each other.

B. Bag of Words

In tests carried out with BoW and a window with just one command, reasonably low accuracy rates were observed, ranging from 89 to 92 %, with emphasis to Neural Networks, as shown in Figure 3, which obtained 92,090 % accuracy with Random Oversampling.

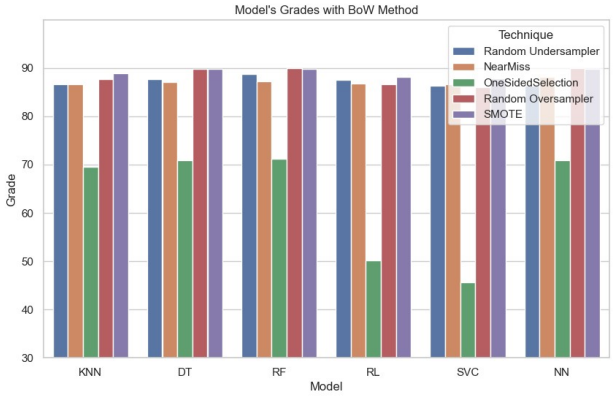

Figure 3. Model's Grades with BoW method.

The most unfavorable results were observed when using the Random Undersampling method, which, in all tested algorithms, recorded an accuracy of less than 90%, apart from the Random Forest algorithm.

These results, in general, demonstrate the limitations of this approach in effectively differentiating between benign and malicious commands when analyzed in isolation.

C. TF-IDF

In this approach, context analysis was carried out through the window of *N* commands, in this case 3 commands.

This window brought greater substance of information to the context of the command, with it was possible to achieve 94,338 % accuracy training with Random Forest and balanced with SMOTE, as shown in Figure 4.

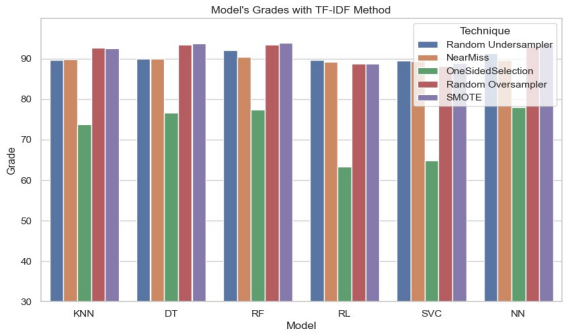

Figure 4. Model's Grades with TF-IDF Method.

D. Doc2Vec

This implementation used the same window with 3 commands mentioned previously. The results obtained in training with Random Forest and balancing with Random Oversampling reached a 'grade' of 99,999 %, as can be seen in Figure 5.

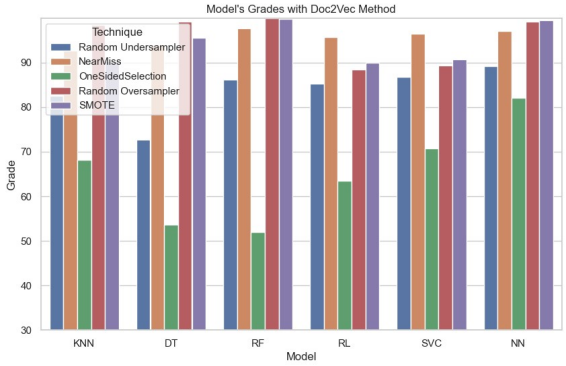

Figure 5. Model's Grades with Doc2Vec Method.

E. One Side Selection

In tests performed, this balancing method consistently achieved high accuracy compared to other algorithms. However, it was notable that their F1-Score and MCC values remained significantly low, as illustrated in Figure 6.

Obtaining a low F1-Score implies that, although the model demonstrates good accuracy in predicting the majority class, it fails to properly identify the minority class effectively. This observation suggests a limitation of the model in dealing with the imbalance between classes, which may compromise its applicability in intrusion detection scenarios. In these scenarios, malicious inputs often deviate from normal patterns and constitute a minority class, requiring accurate detection for effective security.

V. CONCLUSIONS

Considering that the nature of LOLBins attacks rely on legitimate operating system binaries, the need for a contextual analysis of the commands executed is clear. The initial study with Bag of Words, which was based on the evaluation of isolated lines of command, proved to be a considerable challenge, resulting in lower scores and low effectiveness. However, when adapting the approach to classification based on a command window, thus representing the context in which the commands are executed, a significant improvement in performance metrics was observed.

Finally, the results of the tests conducted revealed that the techniques focused on increasing the minority class, in combination with the Doc2Vec technique, showed promising results. This observation is evidenced in Table III, where the five best evaluations were achieved using Doc2Vec. Regarding machine learning algorithms, Decision Trees (DT) and Random Forest (RF), along with neural networks, have demonstrated superior performance, not only with Doc2Vec, but also with other NLP techniques.

TABLE III. BEST 5 GRADES OBTAINED IN THE TESTS

Technique	***Model***	***Method***	***Grade (%)***	***MCC***
ROS	RF	Doc2Vec	99.99%	99.99
SMOTE	RF		99.76%	99.61
SMOTE	NN		99.55%	99.29
ROS	DT		99.19%	98.71
ROS	NN		99.14%	98.62

The results of these tests emphasize that the best effectiveness of the proposed models is achieved through a synergy between NLP techniques and artificial data balancing, complemented by the selection of appropriate machine learning algorithms.

REFERENCES

[1] LIVING OFF THE LAND (LOTL) ATTACKS AND DEFENDING AGAINST APT'S, April 2023. Infosec News. Accessed: 2023-12-05.

[2] Anas Rulloh Alamsyah, S Rahma, Nadira Sri Belinda, and Adi Setiawan. Smote and nearmiss methods for disease classification with unbalanced data case study: Ifls 5. volume 2021, 01 2022.

[3] Tiberiu Boros, Andrei Cotaie, Antrei Stan, Kumar Vikramjeet, Vivek Malik, and Joseph Davidson. Machine learning and feature engineering for detecting living off the land attacks, 2022.

[4] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):1–13, 2020.

[5] Davide Chicco, Niklas Totsch, and Giuseppe Jurman.¨ The matthews correlation coefficient (mcc) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioData mining*, 14(1):1–22, 2021.

[6] Kateryna Chumachenko. Machine learning methods for malware detection and classification. 2017.

[7] Kuiye Ding, Shuhui Zhang, Feifei Yu, and Guangqi Liu. Lolwtc: A deep learning approach for detecting living off the land attacks. In *2023 IEEE 9th International Conference on Cloud Computing and Intelligent Systems (CCIS)*, pages 176–181, Aug 2023.

[8] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.

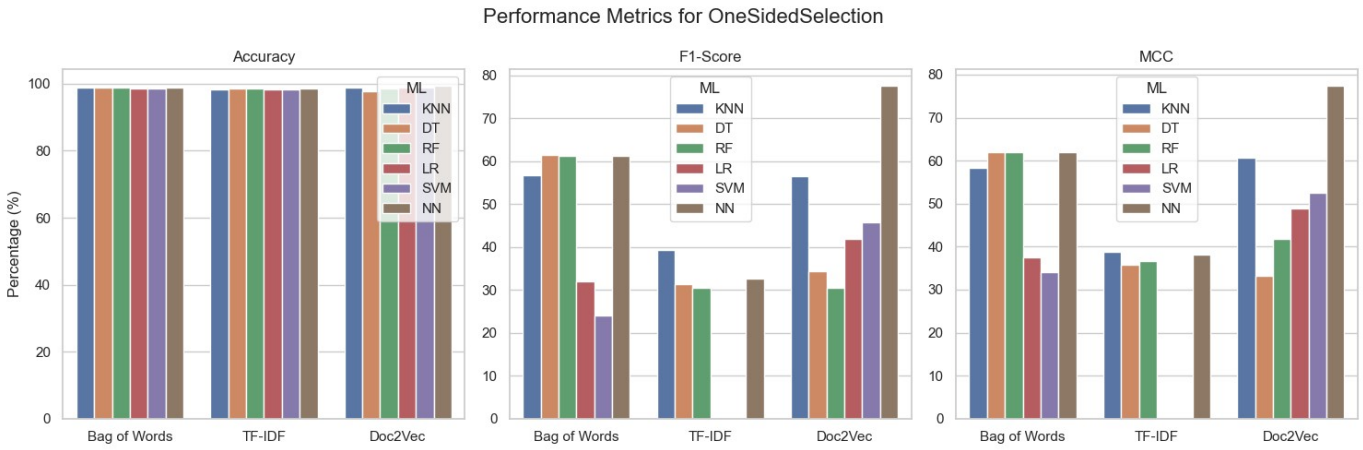

Figure 6. OneSideSelection's Metrics.

[9] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. Handling imbalanced datasets: A review. *GESTS international transactions on computer science and engineering*, 30(1):25–36, 2006.

[10] M. Kubat. Addressing the curse of imbalanced training sets: One-sided selection. *Fourteenth International Conference on Machine Learning*, 06 2000.

[11] Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv eprints*, page arXiv:1607.05368, July 2016.

[12] Mandiant. Barracuda esg zero-day vulnerability (cve-2023-2868) exploited globally by aggressive and skilled actor, suspected links to china, 2023. Accessed: 2024-01-24.

[13] National Institute of Standards and Technology (NIST). CVE-20232868: Barracuda Networks ESG Appliance Improper Input Validation Vulnerability. https://nvd.nist.gov/vuln/detail/CVE-2023-2868, 2023. Accessed: 2024-01-24.

[14] D. Thuy Ngan. Classification of linux commands in ssh session by risk levels. Master's thesis, University of Namur, Faculty of Computer Science, September 2020. Student thesis: Master in Computer Science with Professional focus in Data Science.

[15] Talha Ongun, Jack W. Stokes, Jonathan Bar Or, Ke Tian, Farid Tajaddodianfar, Joshua Neil, Christian Seifert, Alina Oprea, and John C. Platt. Living-off-the-land command detection using active learning. *In Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*, RAID '21, page 442–455, New York, NY, USA, 10 2021. Association for Computing Machinery.

[16] Emilio Pinna and Andrea Cardaci. GTFOBins, 2023. Accessed: 2023-12-05.

[17] Hemant Rathore, Swati Agarwal, Sanjay K Sahay, and Mohit Sewak. Malware detection using machine learning and deep learning. In *Big* Data Analytics: 6th International Conference, BDA 2018, Warangal, India, December 18–21, 2018, Proceedings 6, pages 402–411. Springer, 2018.

[18] Guilherme Silveira and Bennett Bullock. *Machine Learning – Introdução a Classificação* . Editora Casa do Codigo, São Paulo, Brasil, 2020.

[19] Gary Weiss. *Foundations of Imbalanced Learning*, pages 13–41. 06 2013.