Machine learning approach to flow-based brute-force attacks detection

Simone Mattia
CybergOn
University of Milano-Bicocca

Abstract— **In computer security area, anomaly detection algorithms are a very powerful tool for detecting different types of attacks, especially in case of zero-day [1] exploits and encrypted/obfuscated data where signature-based systems are not effective.**
One of their principal applications is related to network traffic analysis, where several categories of attacks stand out, including brute-force attacks. The brute-force attacks detection is becoming more difficult due to the use of more sophisticated techniques that distance, both temporally and geographically, the individual attempts from each other [2]. Considering this, in order to reduce the number of false negatives, one solution could be to analyse the traffic at the level of each individual TCP flow.
To validate this hypothesis i designed a machine learning model that allows to distinguish legitimate RDP [3] traffic from brute-force attempts analyzing each individual TCP flow in terms of packet and session based features.

I. Introduction

Remote Desktop Protocol (RDP) is a proprietary protocol developed by Microsoft that provides users to authenticate to another computer via a network connection: if successful, full access is provided to the target machine, so it can be an optimal initial attack vector.
Although the impact of the account lockout policies [4], which determines the number of failed sign-in attempts that will cause a user account to be locked, RDP bruteforces are still present using different techniques: use of a large pool of geographically distant IPs and the rotation of username-password combinations. These techniques allow an attacker to try the same username from the same IP a few times over an extended period of time, while maintaining the same number of attempts. Another key factor is that the RDP protocol encrypts the traffic, so we cannot analyze the content of the communication, we can only limit ourselves to the physical characteristics of the traffic.
Considering evasion techniques and protocol encryption, in the context of network traffic analysis, one solution is to learn the physical differences, in terms of packet and session based features, between malicious and normal TCP connection: this task can be done optimally using machine learning [5].

II. Dataset

The data were acquired using NFStream [6], a flexible network data analysis framework in Python, that allow capturing the traffic of the entire enterprise network by aggregating it by connection flows and extracting metrics for each of them.
Two datasets were populated:

- normal data, 100000 rows: RDP traffic related to the normal use of the protocol in the enterprise network;
- anomaly data, 41631 rows: RDP traffic related to a series of bruteforce RDP attacks, from internal and public IPs directed to different hosts on the enterprise network, generated using the main tools available in the distributions used to perform penetration testing (crowbar [7], hydra [8] and patator [9]).

Both datasets have the same features and each row refers to an individual TCP flow.
NFStream automatically extract 65 features [10] from network traffic, summarizing:

- general features: source/destination/protocol informations (ip, port, vlan, high-level protocol, etc.);
- time related features: first/last packet timestamp in ms (both bidirectional, src2dst and dst2src), flow duration in ms, mean/minimum/maximum/standard-deviation time between packets in ms (both bidirectional, src2dst and dst2src);
- packet size features: transmitted bytes (both bdirectional, src2dst and dst2src), mean/minimum/ maximum/standard-deviation in packet size in bytes (both bidirectional, src2dst and dst2src);
- tcp flag features: number of packet with syn/cwr/ ece/ack/psh/rst/fin tcp flag (both bidirectional, src2dst and dst2src).

III. Preprocessing

A. Handling missing and invalid rows

To improve data quality, it was chosen to remove all rows containing null or invalid fields: no missing data were present in the two datasets, but there were some invalid data.
Specifically, rows related to IP reported as malicious on the AbuseIPDB platform [11] were considered invalid: this was done since it was not possible to understand whether the traffic related to those rows was malicious or not.

B. Data labeling

In both datasets a new field called ”label” was inserted to indicate whether the traffic was legitimate or not: were set as 0 (normal) all rows in the first dataset while in the second only those with source IPs and destinations different from those

from which the simulated attacks were carried out, while the remaining ones were labeled with 1 (anomaly).
Once labeled, it was possible to merge the two datasets into a single one of 140675 rows.

C. Feature selection

A first manual feature selection was carried out to remove all fields not needed for analysis:

- all the features previously indicated as general (source ip, source port, protocol, destination ip, etc.) have been removed since they do not contain information useful to the problem;
- all the features previously indicated as time related, except for flow duration, have been removed since they can be not useful (e.g. timestamp of the last dst2src packet in ms) or too influenced by the distance between the two hosts in the network (e.g. maximum time between dst2src packets in ms) causing errors in the estimates.

Thus remaining with 41 features.

D. Variable transformation

The only transformation operation performed on the features was normalization. Data normalization can normalize the traffic data or feature value to a range of zero to one or negative one to positive one so as to reduce data redundancy, enhancing the integrity and efficiency of model training [5].

E. Partitioning

A partition of the dataset into three disjointed parts was carried out, respectively:

- 80% (112539 rows) training dataset: set of data that is used to train and make the model learn the hidden features/patterns in the data;
- 10% (14068 rows) validation dataset: set of data, separate from the training set, that is used to validate our model performance during training (hyperparameters tuning and feature selection);
- 10% (14068 rows) test dataset: set of data used to test the model after completing the training, it provides an unbiased final model performance metric.

IV. Training

A. Model

Random Forest was chosen as the classification model, the choice was motivated by the following considerations:

- in the area of network traffic analysis it has performed well, in term of accuracy, with structured data such as packet and session based features [5];
- with enough trees it's robust against overfitting [12].

B. Hyperparameters tuning

The main hyperparameters required by the model are [13]:

- n_estimators: the number of trees in the forest;
- criterion: the function to measure the quality of a split;
- max_features: the number of features to consider when looking for the best split.

An exhaustive grid search [14] was used to select the best combination between the following parameters:

- n_estimators: 25, 50, 75, 100, 150, 200, 250, 300;
- criterion: gini, entropy, log_loss;
- max_features: sqrt, log2.

The algorithm was performed on the validation set and selected the combination of hyperparameters that provided the highest F_1 score on a 10-fold cross-validation, the F_1 metric was chosen due to the imbalance of the dataset.
Selected hyperparameters: 100 (n_estimators), gini (criterion) and log2 (max_features).

C. Feature selection

To reduce the dimensionality of the dataset a recursive feature elimination [15] was carried out on the validation set to selected the minimal set of feature that provided the highest f_1 score on a 10-fold cross-validation.
Fig. 1 shows the change in performance according to the number of features: the optimal number of features is 15; then fig. 2 shows the selected features with their relative importance in prediction.

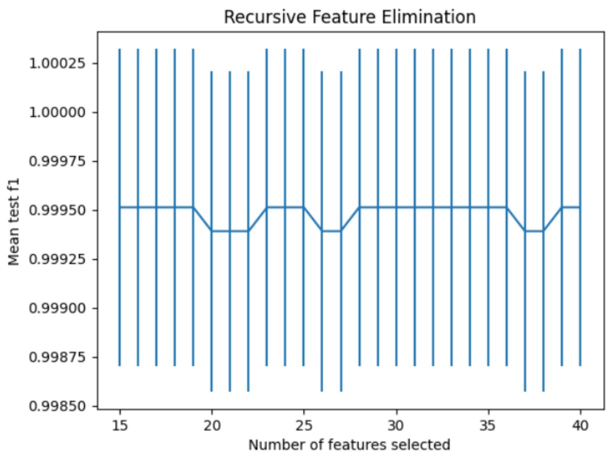

Fig. 1. Recursive Feature Elimination

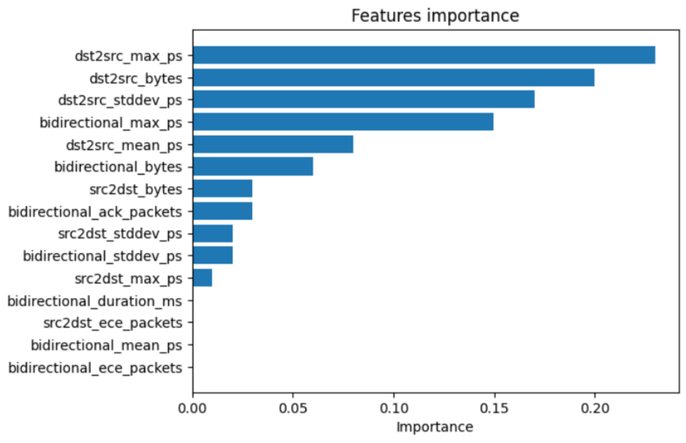

Fig. 2. Feature Importance

V. Evaluation

A. Metrics

Several metrics based on true-positive (TP), true-negative (TN), false-positive (FP) and false-negative (FN) counts were used to evaluate the performance of the model:

1) precision: fraction of records that actually turns out to be positive in the group the classification model has declared as a positive class;

$$precision = \frac{TP}{TP + FP}$$

2) recall: fraction of positive records correctly predicted by the classification model

$$recall = \frac{TP}{TP + FN}$$

3) F_1: metric summarizing precision and recall;

$$F_1 = \frac{2 * recall * precision}{recall + precision}$$

4) balanced accuracy: is the macro-average of recall scores per class.

$$balanced - accuracy = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right)$$

In comparison with accuracy, the balanced version avoids inflated performance estimates on imbalanced datasets [16].

B. Holdout

After training the model, with previously selected hyperparameters and features, an evaluation was carried out on the predictions made on the previously partitioned testing dataset, with unseen data.
The model obtained the following results:

- precision: 1.0;
- recall: 0.9995;
- F_1: 0.9998;
- balanced accuracy: 0.9997.

C. Cross validation

Finally, the model, with previously selected hyperparameters and features, was evaluated using cross-validation.
Specifically 10-fold cross-validation was used: the entire dataset was divided into ten subsets and the models were trained ten times, using, at each iteration, nine parts as the training set and one part as the test set, calculating the performance at each iteration.
As shown by fig. 3 the results of the 10-fold cross-validation are aligned with the holdout, arithmetic mean of performance:

- precision: 0.9993;
- recall: 0.9999;
- F_1: 0.9996;
- balanced accuracy: 0.9998.

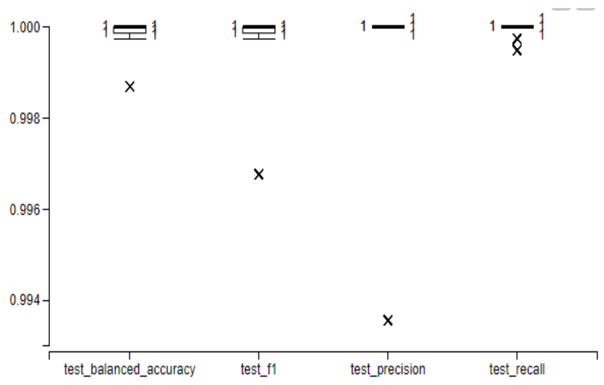

Fig. 3. Cross-Validation Boxplot

D. Conclusion

In conclusion, the model possesses excellent performance in terms of recall (0.9999), means that very few positive records were misclassified as negative class. It also has an excellent precision value (0.9993), which means that very few negative records were misclassified as positive.
As a result, analyzing the packet and session based characteristics of each individual TCP flow turns out to be an excellent strategy for bruteforce attack detection.
In addition, the same workflow could be applied to design new models to detect bruteforces on other protocols or similar attack categories (e.g.web application fuzzing).

References

[1] https://en.wikipedia.org/wiki/Zero-day_(computing)
[2] Martin Drasar, Jan Vykopa and Philipp Winter. Flow-based Brute-force Attack Detection.
[3] https://en.wikipedia.org/wiki/Remote_Desktop_Protocol
[4] https://learn.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/account-lockout-threshold
[5] Zihao Wang, Kar-Wai Fok, Vrizlynn L. L. Thing. Machine Learning for Encrypted Malicious Traffic Detection: Approaches, Datasets and Comparative Study.
[6] https://www.nfstream.org/
[7] https://github.com/galkan/crowbar
[8] https://github.com/vanhauser-thc/thc-hydra
[9] https://github.com/lanjelot/patator
[10] https://www.nfstream.org/docs/api
[11] https://www.abuseipdb.com/
[12] https://builtin.com/data-science/random-forest-algorithm
[13] https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[14] https://scikit-learn.org/stable/modules/grid_search.html#grid-search
[15] https://scikit-learn.org/stable/modules/feature_selection.html#rfe
[16] https://scikit-learn.org/stable/modules/model_evaluation.html#balanced-accuracy-score