CS4035 Cyber Data Analytics - Assigment 3

Andrei Simion-Constantinescu (4915739)
Mihai Bogdan Voicescu (4951433)
Group 66

June 2019

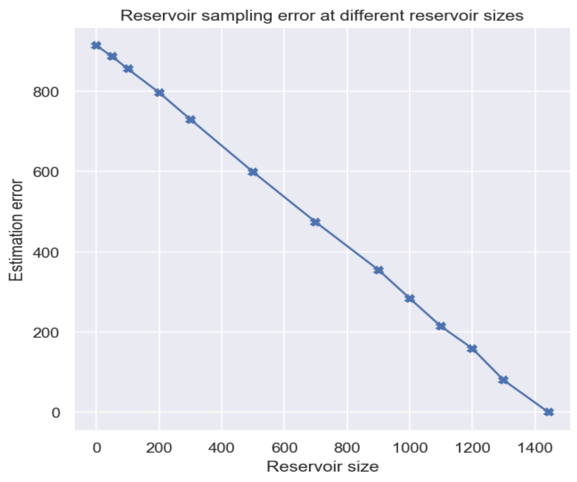
1 Introduction

This is the third assignment in the Cyber Data Analytics course. It is focused on fingerprinting and pattern detection and how this can be used to spot malicous behavior. Our code can be find on Github. Follow the instructions in README.md for setting up the environment and understanding the code structure.

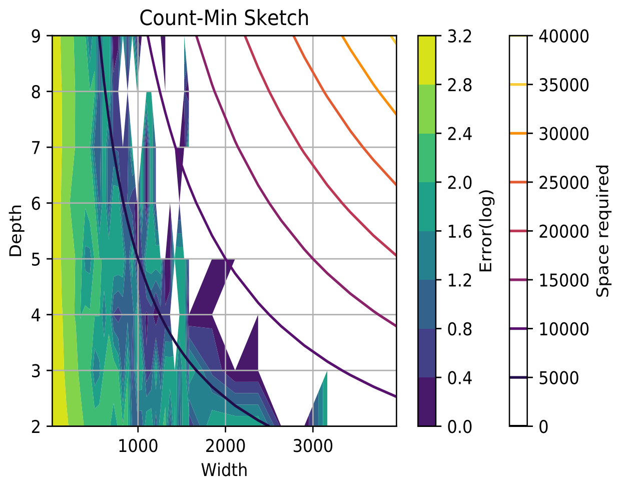
2 Sampling task

For the first two tasks, we chose to work with data from Scenario 6. The 10 most frequent addresses the infected host (147.32.84.165) connects with displayed with their frequencies are: (91.212.135.158: 404), (64.59.134.8: 194), (24.71.223.11: 160), (216.32.180.22: 154), (90.177.113.3: 100), (147.32.96.45: 100), (65.55.37.72: 96), (65.55.37.88: 94), (65.55.92.136: 93), (65.55.92.168: 90). Reservoir sampling was implemented via Order Sampling using a priority queue to store the smallest k-elements (size of reservoir) according to their randomly uniform generated tag. The priority queue content is the reservoir data used to estimate the stream distribution. We used a range of reservoir sizes from 50 to the number of flows that belongs to the infected hosts, 5877. The estimated distribution when using different reservoir sizes can be found in `logs/2019-06-20_09_22_43_log0.html`. As an example, when using a reservoir of size 5000, the top 10 estimated distribution is: (91.212.135.158: 334), (64.59.134.8: 174), (24.71.223.11: 140), (216.32.180.22: 129), (90.177.113.3: 92), (147.32.96.45: 83), (65.55.92.136: 82), (202.108.252.141: 80), (65.55.37.88: 79), (65.55.37.72: 78).

The estimation error is determined starting from a maximum value equal to the sum of frequencies of the top-10 IP addresses, calculated for each estimation as the difference between estimated value and real value. In case of one of the top-10 ip addresses not present in the estimation, its whole frequency is added to the error. The accuracy of the algorithm is dependent on the size of the reservoir as shown is Figure 1a, having a linear dependency. This is normal because the distribution of the elements in the queue will start to converge to the real value of the distribution due to the probability of occurrences of an item in the queue being directly proportional to the frequency. The resolution is also directly proportional to the reservoir size. A thing to note is that the error reaches 0 when the reservoir size is almost equal to the number of elements, but using this value defies it's purpose.



(a) Estimation error based on reservoir size



(b) Error and space required depending on width and height. No color means 0

Figure 1: Error evolution for reservoir sampling and count-min sketch for different parameters

3 Sketching task

The Count-min sketch is another probabilistic data structure used to estimate the exact frequency of a certain element in a data stream. The algorithm makes use of multiple hashes, each with it's associated bin. At insert, each element hashed using all the hash functions, resulting in a set of numbers that will be used as indexes for the corresponding bins. Each index is incremented in the hash's bin. At query time, the same hashes are applied and the counts of the indexes from each bin is extracted, resulting in a set of frequencies, out of which we select the smallest. The data structure is characterized by a width and a depth which define the number of hashes and the size of the bins, effectively being stored as a matrix.

Figure 1b shows that the structure can be successfully used to accurately predict the top 10 values and we notice that increasing the width is more efficient in terms of space complexity than increasing the depth.

4 Sampling comparison

Both methods present pretty similar results by fine tuning parameters if the size is decent. If the space is big enough they will present the exact results. Reservoir sampling will underestimate results while Sketch will overestimate results, due to collisions in the hash.

When comparing in terms of time complexity, Reservoir Sampling has $\mathcal{O}(n \cdot \log k)$ and Count-min Sketch $\mathcal{O}(n \cdot d)$, where n is the size of the stream, k the reservoir size and d the depth. When looking at space complexity, we define size as the *reservoir size* for Reservoir and $width \cdot depth$ for Sketch and display a couple of top 100 errors at same size in Table 1. If a ranking is required, Reservoir sampling is better suited. If a distribution is required then both can be used, but if an exact count is desired Sketch should be used.

Method	Size	Error
Reservoir	500	370.66
Reservoir	1000	335.28
Sketch	500	531.48
Sketch	1000	182.30

Table 1: Comparison of average 50-runs top-100 error between Reservoir and Sketch in terms of size

5 Flow data discretization task

In order to detect the fingerprints we have to discretize the netflow information, but we must first settle on what to select from it. If we take a look at Figure 2, we notice that the distribution of flags is very different for infected hosts, compared to normal ones. The infected host is sending more ICMP packets and their associated flags. Because the protocol can be implied from the flags being not a single overlap of flags between different protocols, we decided to use the flags as our first feature.

A host has a limited bandwidth, there is no way around this, as such a malicious software wants to maximize the band width usage in case of a Distributed Denial-of-Service (DDoS), generating as many requests as possible, ending up with requests with a small sizes. If we check Figure 3, we notices that this is true. Additional visualization where performed for packets, duration and flags for the chosen infected host (147.32.84.165) and can be seen in an additional notebook, `Flow visualization.ipynb`. As displayed in the notebook, packets feature displays the same trend as bytes (Figure 8 from Appendix), therefore the second selected features is bytes.

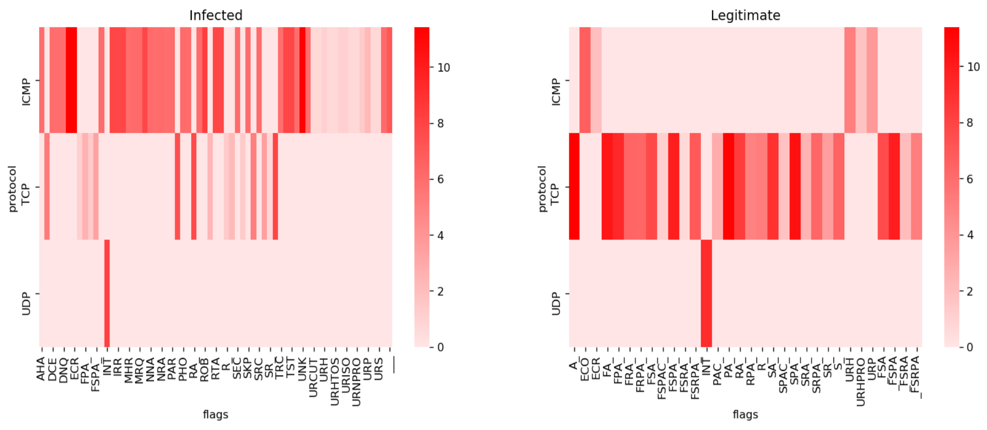


Figure 2: Heat-map of flags/protocol distribution

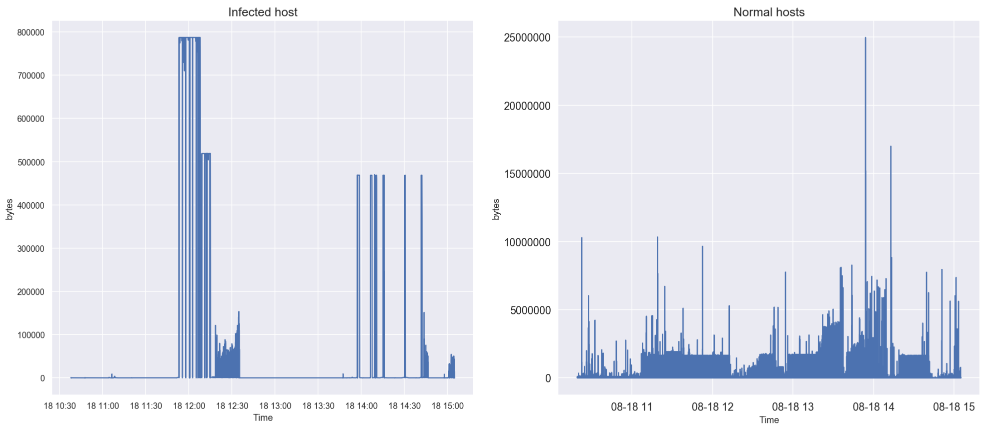


Figure 3: Number of bytes sent by individual requests on an infected compared to normal hosts. Note that the y-range of the two subplots is different

After selecting flags and bytes as the feature to be discretize, we applied the ELBOW method to select the optimal number of bins using KMeans clusterization method [3]. By visual inspection of Figure 7 from Appendix, the optimal number of bins is determined as 5. Following the approach from [5], we used percentiles method to discretize bytes (being a numerical feature) having [20th, 40th, 60th, 80th] as percentiles (5 bins optimal number) and simple conversion to integer codes for flags (categorical feature). The combination of the two discretize features is done using the formula proposed by [5].

6 Botnet profiling task

For this task we chose to use a N-grams model. We use the previously discretized values to compute the possible states. We have 78 possible flags and 5 different buckets for the bytes size. This amounts to 390 possible states(maxSize) in which we will place each request. Because we also want to incorporate the frequency of requests in our model we use a sliding window of 2 seconds. This will result in requests being processed N + 1 times, where N is the number of requests that follow the request within a 2 seconds period, due to how pandas.DataFrame#rolling works. We extend the states with 3_ngrams by mapping them with the following formula:

$$N\_GRAM = State_1 * maxSize^2 + State_2 * maxSize + State_3$$

This will return a final state from range 0 to $maxSize^3$. We do this for all the requests that have the source IP the desired target and count how many times each state occurs, obtaining a histogram which will represent the fingerprint. If we take a look at Figure 4 we notice there are some huge spikes in certain states.

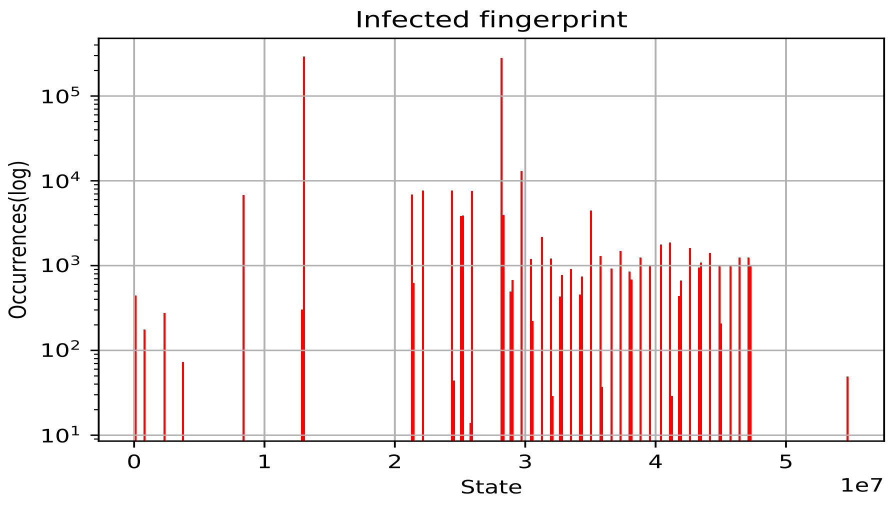


Figure 4: Ngrams histogram of an infected PC on a log scale

If we do the same for normal PCs and plot them against each other we notice that they have quite a different fingerprint, as shown is Figure 5. In order to compare 2 fingerprints for similarity we chose to use the mean squared error metric.

Computing the fingerprint and sorting by MSE we notice that the top 10 results are the infected computers, proving that this method is highly effective. Even more the difference between the infected and non-infected computers

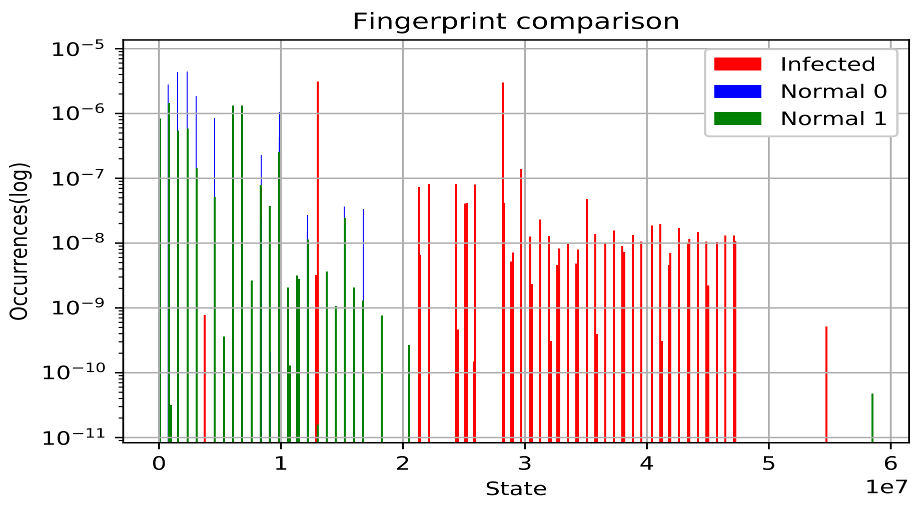


Figure 5: Ngrams histogram of an infected PC and 2 normal ones on a log scale

is huge, as shown in Figure 6, the least similar infected computer presenting a 0.0002 MSE compared to 0.0558, the most similar non-infected, almost 280 times bigger.

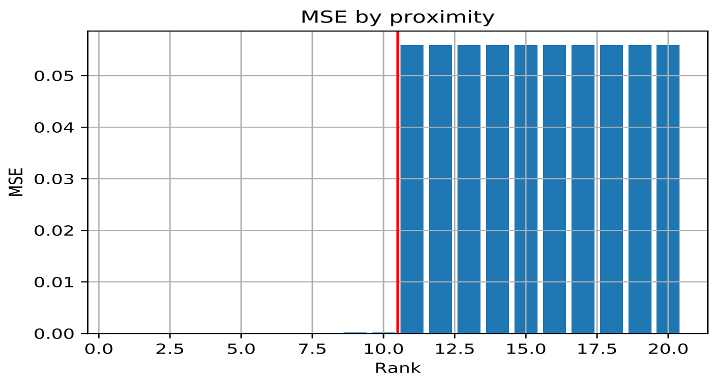


Figure 6: MSE of the first 20 entries sorted by MSE. The red line marks the end of the infected computers. We notice the infected computers are barley visible.

7 Flow classification task

In order to build a classifier that is trained on individual NetFlows, we decide to use only the following relevant information available at each row: duration, protocol, flags, packets and bytes. For the categorical features protocol and flags, we convert them into integer codes based on the unique values for each. We eliminate the background flows and convert the labels as follows: LEGITIMATE as 0 and Botnet as 1. In order to assure the same number of infected host and normal hosts in training and testing data, we split the data using this approach: from the total of the 10 infected hosts, we separate the infected hosts as infected_train being all the netflows from the first 5 infected hosts and infected_test from the last 5; for the normal netflows, we performed a random train/test split of 0.79/0.21 obtaining normal_train and normal_test; for getting the training dataset, infected_train was concatenate with normal_train and shuffle, same for the test dataset using infected_test and normal_test. After the preprocessing pipeline, we ended with a training dataset of 443237 rows and testing dataset of 202121, the training data label distribution being 0: 254179, 1: 189058. The training data is characterized by negative skeweness, the ratio between normal and infected netflows being approximately 1.3. To tackle the class imbalance issue [4], we decided to use sample weighting using 1.3 for botnet flows and 1 for the rest. Being inspired by the popularity of using Random Forest as a classifier for detecting anomalous behaviours in netflows [1], we chosen it and tune the hyperparamets to obtain our final model which is summarize in Table 2.

Classifier	N_estimators	min_samples_split	min_samples_leaf	max_depth	criterion
Random Forest	200	4	2	9	gini

Table 2: Random Forest Classifier hyperparameters

We evaluated the performance of our proposed classifier on the created testing dataset on packet level [2] looking at the probability generated for each testing row and on host level [5] looking at the list of unique ip addresses and classify a host as being infected is at least one netflow from its traffic has a probability higher than the classification threshold. We used a range of values for the classification threshold and found out that the model confidence is very high being able to perform the best when taking into account metrics for both packet and host level with a threshold of 0.95. The main results are displayed in Table 3 and the full ones can be found32. be opening the file generated by the logging system during training and testing, `logs\2019-06-21_20_24_19_log0.html`.

	Packet level				Host level			
Threshold	FP	TP	Precision	Recall	FP	TP	Precision	Recall
0.5	42	132592	0.9997	0.9867	39	5	0.1136	1.0000
0.7	32	132224	0.9998	0.9839	27	5	0.1562	1.0000
0.9	31	132177	0.9998	0.9836	24	5	0.1724	1.0000
0.95	0	131586	1.0000	0.9792	1	5	0.8333	1.0000
0.99	0	131480	1.0000	0.9784	1	5	0.8333	1.0000
0.999	0	20194	1.0000	0.1503	1	5	0.8333	1.0000

Table 3: Results of final classifier evaluated on packet and host level

8 Bonus

For the bonus we have read the offered blog post and took inspiration from a previous research done on exploring the effects of adversarial data on Botnet classification using Random Forest [1]. We have therefore selected to alter the number of packets, bytes and duration from each individual package. All these operations are plausible and easy to be applied by an attacker without altering the logic of the bot (e.g increase flow duration by introducing a latency). All of the perturbations values are in a realistic range. If we take a look at the Table 4, we notice that the alterations are successful only on individual packet level, proving useless at host level because some flows will still appear as infected, causing the whole host to be marked as infected.

If we try the same method with the Profiling, we notice that no infected hosts are marked as normal. This is because altering the values with a constant number does not alter what percentile bucket the request lands in, effectively causing no change in the registered state. If we spice things up a bit and add random values on each request, the host amazingly is still registered as infected. This came as a big surprise for us meaning the added values are not big enough to actually register a change or that the usage of the flags is more revealing then the bit size.

			Packet level			Host level		
Packets	Bytes	Duration	TP	Precision	Recall	TP	Precision	Recall
0	0	0	131586	1.0000	0.9792	5	0.8333	1.0000
0	0	1	21096	1.0000	0.1570	5	0.8333	1.0000
1	0	0	21112	1.0000	0.1571	5	0.8333	1.0000
0	1	0	21114	1.0000	0.1571	5	0.8333	1.0000
1	1	1	21096	1.0000	0.1570	5	0.8333	1.0000
10	16	10	21084	1.0000	0.1569	5	0.8333	1.0000
30	256	45	21084	1.0000	0.1569	5	0.8333	1.0000
100	1024	120	20922	1.0000	0.1557	5	0.8333	1.0000
0	0	120	21084	1.0000	0.1569	5	0.8333	1.0000
100	0	0	20240	1.0000	0.1506	5	0.8333	1.0000
0	1024	0	21114	1.0000	0.1571	5	0.8333	1.0000

Table 4: Results of adversarial search the classifier

9 Conclusion

Both the classifier and the profiler proved to be extremely efficient in detecting similarly infected hosts. Though our current version of profiler has a big complexity due to the 2 seconds repeat window, we are sure it can be lowered remaining still useful. Unfortunately since none failed or presented significantly better results than the other, we can just speculate about which is better. A big advantage of the profiler is that it's complexity is almost the same for scanning for multiple footprints. The fingerprint is built once and analyzed against any amount of registered infected fingerprints for similarities, while the classifier should require multiple models, one for each variation of the virus.

10 Appendix

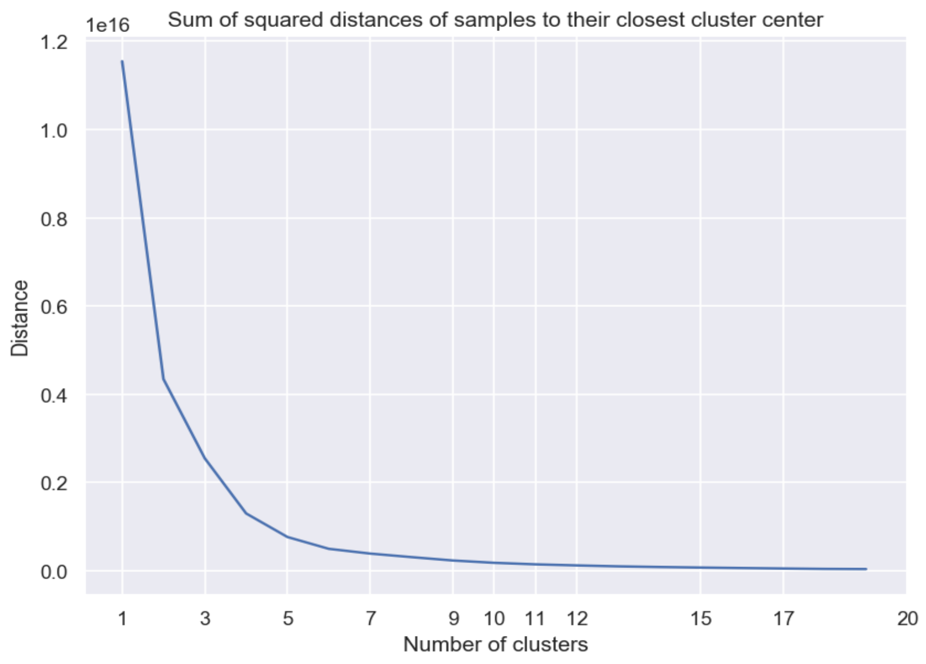


Figure 7: Elbow plot

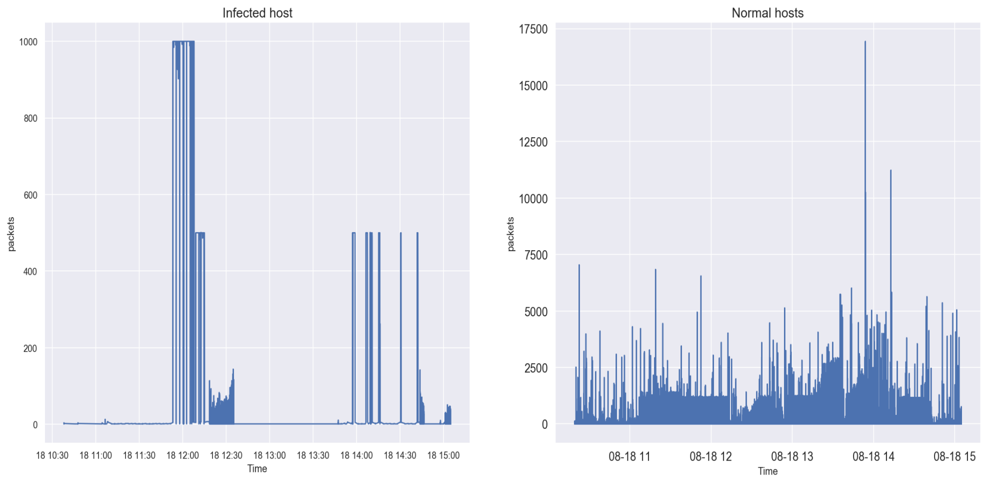


Figure 8: Number of packets sent by individual requests on an infected compared to normal hosts. Note that the y-range of the two subplots is different

References

[1] Giovanni Apruzzese and Michele Colajanni. Evading botnet detectors based on flows and random forest with adversarial samples. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[2] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *computers & security*, 45:100–123, 2014.

[3] Cyril Goutte, Peter Toft, Egill Rostrup, Finn Å Nielsen, and Lars Kai Hansen. On clustering fmri time series. *NeuroImage*, 9(3):298–310, 1999.

[4] Guo Haixiang, Li Yijing, Jennifer Shang, Gu Mingyun, Huang Yuanyue, and Gong Bing. Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, 73:220–239, 2017.

[5] Gaetano Pellegrino, Qin Lin, Christian Hammerschmidt, and Sicco Verwer. Learning behavioral fingerprints from netflows using timed automata. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 308–316. IEEE, 2017.