Assignment 3

Group 7

June 21, 2018

Sampling task

The sampling task was tested on Scenario 2 https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-43/. The infected host, we want to investigate with whom it connects with, is '147.32.84.165'. In table 1 we can see the 10 most frequent addresses the host connects with.

IP	# of connections	percentage
93.23.181.44	6442	12.4%
174.128.246.102	4101	7.9%
174.37.196.55	3707	7.1%
173.236.31.226	3410	6.6%
184.154.89.154	3344	6.5%
67.19.72.206	3135	6.0%
72.20.15.61	2966	5.7%
46.4.36.120	1627	3.1%
212.117.171.138	967	1.8%
147.32.80.9	787	1.5%

Table 1: The IPs that connect frequently with the selected infected host.

Next we implemented the MIN-WISE sampling for many different reservoir sizes. The dataset, after the preprocessing we did in order to keep only the rows that contained the selected infected host, has 51809 rows. We checked the algorithm for reservoir sizes in range 100-10000 with step 100. The detailed results can be seen in the jupyter notebook, and here we will present some indicative results.

As we can observe in table 2, already for reservoir size equal to 100, 8 out of 10 most frequent ips are found. However we can see that the ground truth 1 has big differences with these results, thus we increase the reservoir size. With k=200 again there are two ips that should have appears but they didn't. With reservoir size = 500 all top 10 ips appear, but in different order. As expected the higher the reservoir size the better the approximation of the most frequent ips connected with the host, however the most exhaustive the computations. For k=9700 we get exactly the same ips and percentages of appearance close to the ground truth. We can conclude that we need a sample less than 20% of the original dataset in order to approximate the top 10 most frequent ips with high accuracy, however we can not get the exact number of connections (only if we set the reservoir size to the original size of the data, which of course does not make any sense).

	k=100		**k=200**		**k=500**	
IP	# connections	percentage	# connections	percentage	# connections	percentage
93.23.181.44	11	11%	23	11.5%	66	13.2%
174.128.246.102	13	13%	23	11.5%	42	8.4%
174.37.196.55	7	7%	14	7%	40	8.0%
173.236.31.226	7	7%	11	5.5%	25	5.0%
184.154.89.154	7	7%	17	8.5%	39	7.8%
67.19.72.206	3	3%	5	2.5%	19	3.8%
72.20.15.61	3	3%	9	4.5%	28	5.6%
46.4.36.120	-	-	-	-	14	2.8%
212.117.171.138	2	2%	4	2.5%	8	1.6%
147.32.80.9	-	-	-	-	8	1.6%

	k=1000		**k=2000**		**k=10000**	
IP	# connections	percentage	# connections	percentage	# connections	percentage
93.23.181.44	119	11.9%	244	12.2%	1272	12.7%
174.128.246.102	77	7.7%	168	8.4%	785	7.85%
174.37.196.55	87	8.7%	131	6.55%	736	7.36%
173.236.31.226	56	5.6%	151	7.55%	665	6.65%
184.154.89.154	76	7.6%	129	6.45%	654	6.54%
67.19.72.206	50	5.0%	128	6.4%	593	5.93%
72.20.15.61	54	5.4%	129	6.45%	560	5.6%
46.4.36.120	34	3.4%	66	3.3%	318	3.18%
212.117.171.138	22	2.2%	33	1.65%	181	1.81%
147.32.80.9	18	1.8%	27	1.35%	157	1.57%

Table 2: The changes in the *true* top 10 frequent ips with different reservoir sizes for min-wise hashing.

Sketching task

For the Count Min sketch we played with different width and heights which were chosen by setting $w = \lceil \frac{e}{\epsilon} \rceil$ and $d = \lceil \ln \frac{1}{\delta} \rceil$. ϵ is the error we accept and δ is the probability of being outside that error. In the following table, we can see how the algorithm performs for specific values of ϵ and δ, all the experiments are presented at the jupyter notebook.

w	d	ϵ	δ	Accuracy
2718	9	0.001	0.0001	100%
2718	5	0.01	0.01	100%
2718	2	0.01	0.1	99.996%
544	9	0.005	0.0001	99.945%
544	7	0.005	0.001	99.936%
544	5	0.005	0.005	99.909%

As we can see the algorithm for specific w, d produces exactly the same results as counting the top 10 most frequent IPs one by one (ground truth). With 2718 counters and 4 hash functions we get the top 10 IPs with accuracy 100%. With the min wise hashing we couldn't get that accurate results, for 100% accuracy we have to get extremely big reservoir size, which is inefficient. We compared the two methods in more detail, as far as the run time is concerned. For the min wise hashing with reservoir size equal to 5000 the code takes 3.949 sec to execute, on the other hand, the count min algorithm just needs 0.3 sec (w= 2718, d=4) and it produces much more accurate results, since we accept error $\epsilon = 0.01$.

As far as the space efficiency is concerned, the count min again outperforms the min wise hashing. Count min needs a matrix with dimensions $w \times d$, which is much less than the number of the elements. Min hashing requires more space, since for each row we have to specify a random number, thus we have as many values as the number of the IPs we want to count, in our case more than 50000.

Botnet flow data discretization task

For this question we use scenario 10 as requested. The first step is to load the dataset and remove the background flows since we don't know if they are botnet or benign. Next we investigate for one infected host which features are more discriminative, in order to represent the botnets. The selected host was "147.32.84.208". As we can see the protocol feature is informative for the distinction between the two classes, since the most commonly used protocol in normal flow is UDP 1, but for the infected host is the ICMP 2.

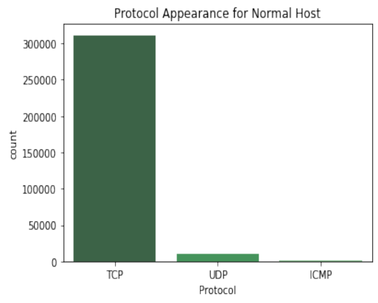

Figure 1: Protocols used for Normal Flows

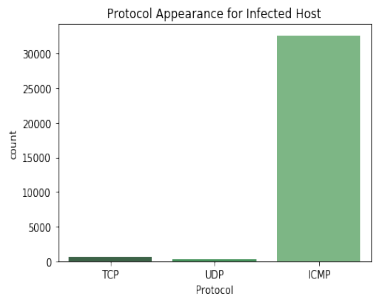

Figure 2: Protocols used for Infected host

We also visualized the features: number of transmitted packets 3 and the bytes transmitted 4. Both the features show difference between the mean amount of packages or bytes transmitted for the normal and the infected hosts. The last column in the following graphs, is about the selected infected host. We chose to show also this bar in the graph in order to understand how representative the selected host is for the whole set of the infected hosts. As we can see in both 3, 4 the selected infected host has the same behavior as the mean infected host of the whole dataset.

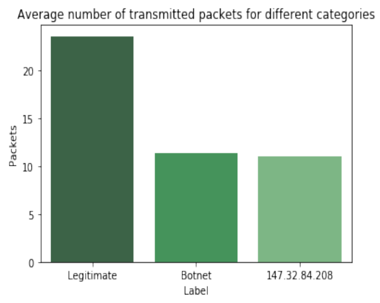

Figure 3: Mean Packages for different Categories

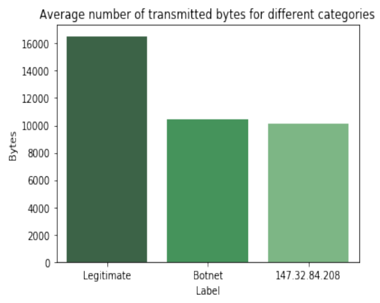

Figure 4: Mean Bytes for different Categories

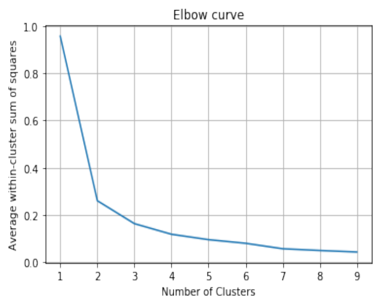

Figure 5: Elbow for Packets

We decided to discretize the protocol and the number of transmitted packages and then combine them as indicated in [1] into a single discrete value. The case of Protocol is easy, since there are only 3 options, TCP, UDP, ICMP. However the Packets feature requires different handling. We first use ELBOW method to find the optimal number of cluster for the values of Packets. As we can see in 5 the optimal number of clusters is 2. Thus we can define the percentiles, which correspond to the ordinal ranks. Then the attribute mapping can be performed. Finally we encode the whole netflow using the discretized columns. After that step we have created a new dataframe which contains the discretized features and the encoding of each row of the initial dataset.

After the discretization we know that some specific codes are most likely to infer to botnets. For example the combination of ICMP protocol's encoding with the encoding of the cluster of low packets transfered (< 10) is highly probable to refer to botnet.

Botnet Profiling task

For this question we will use the discretized data from the previous question. Since we aggregated the data of each host we use sliding window to obtain sequence data. As we present in table 6 we have applied different windows to check the performance of the model. The sequential model we chose is N-grams [2], thus the next step is to extract the n-grams from the aforementioned sequences. We used the top 10 most frequent N-grams in order to model the fingerprints. The fingerprint of each host consists of pairs of N-grams and their corresponding frequency of appearance in the specified host. We compare each fingerprint of the hosts we want to classify with a infected host's fingerprint and a normal host's fingerprint.

For the evaluation of the model we used the procedure of [1]. We will perform a single scenario simulation and multi scenario simulation.

Single Scenario Simulation

In a specific scenario we expect that the malware will be similar. Consequently this is the "easy" case of the testing. For the experiments we chose Scenario 10 https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-51/, which contains 10 infected IPs and 5 normal. One IP of each category will be used as train, which means that we model its fingerprint and we use it as a standard for this category, how we expect a botnet and a normal netflow to look like. The other 15 IPs are used for testing. We classify each testing host based on if its nearest neighbor is the train fingerprint of botnet or normal host.

window	N-grams	TP	FP	TN	FN	Recall	Precision
60ms	2	9	1	4	0	100%	90%
60ms	7	9	2	3	0	100%	81%
90ms	2	9	1	4	0	100%	90%
90ms	7	9	2	3	0	100%	81%

Table 3: N-grams sequential model for different sliding window and N values.

From table 6 we observe that the sliding window does not really affect the results, at least for small differences in time. The most important factor is the number of N-grams we choose. For N=3 we only have one benign IP address classified as botnet. We have to mention that this IP that was misclassified appears just a few times in the dataset, so this result is not really representative. For N=7, two benign IPs are misclassified as botnet. To conclude we can say that the specific model performs well at detecting the botnets, but it also tends to classify some normal host as botnets. It chooses the "safe" way, which means it is more important to have higher recall (the overall goal is to detect all the botnets), than higher precision.

Multi Scenario Simulation

For this simulation we used scenario 10 for training and scenario 9 `https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-50/` for testing. As we did expected the results are worse than the single scenario case. Here 4 benign hosts were classified mistakenly as infected.

window	N-grams	TP	FP	TN	FN	Recall	Precision
60ms	2	10	4	2	0	100%	71%

Table 4: Results of Multi Scenario simulation.

Bonus: flow classification task

For this task we use the same dataset as in the two previous tasks (scenario 10). Our implementation was based on the BClus detection method as proposed in [3]. As proposed in their method, the first step after loading the dataset is to separate the netflows into time windows (apply sliding window). The time window that we used was 2 minutes, since it's enoughtime to capture the botnets behavior without containing to much netflow. Then in each of these windows we applied another 1 minute time window. The reason was that this 1 minute window is enough to capture the botnet synchronization patterns and short enough not to capture too much traffic. In each of these windows we did some aggregations in order to generate new features (to create our new aggregated dataset). Specifically, the netflows were aggregated by unique source IP address. The resulting features on each aggregation were:

- The source IP address
- Amount of unique source ports used by this source IP address
- Amount of unique destination IP address contacted by this source IP address
- Amount of unique destination ports contacted by this source IP address
- Amount of netflows used by this source IP address
- Amount of bytes transfered by this source IP address
- Amount of packets transfered by this source IP address

All the aforementioned aggregated windows are concatenated in order to become our new dataset in which we will apply the chosen classifiers. For that purpose we chose two classifiers, RandomForest and the Decision Tree. This task was splitted into two different classifications (on Packet level and on Host level).

Packet Level

In this part we used the aggregated dataset as it was created in the previous step. Then we split this dataset into 80% training and 20% testing set. At this point we observe that we face class imbalance between the infected and normal classes. To solve that problem we applied SMOTE in the training set, to have equal number of samples from both classes. Then we train the classifier with the training set and test it with the testing one. This process was conducted 10 times and then the average of true positives, false positives, false negative, true negative, precision, recall and accuracy was computed. The results of this classification are presented below.

Classifier	TP	FP	TN	FN	Recall	Precision
RandomForest	211	10	4118	15	93%	95%
Decision Tree	211	15	4113	15	93%	93%

Table 5: Classification results in Packet level.

Host Level

In the host level each host is classified as botnet if it appears in at least one NetFlow which was labeled as botnet. For that reason, we group the aggregated dataset by the source IP addresses. The ground truth of that dataset was extracted from the description of the infected and normal host in the dataset. Then the same procedure was followed as in the packet level in order to make the classification. The results of this classification are the following:

Classifier	TP	FP	TN	FN	Recall	Precision
RandomForest	2	0	102	0	84%	100%
Decision Tree	2	0	102	1	82%	100%

Table 6: Classification results in Host level.

Comparison between sequential model and classifier

From the results of the sequential model and the classifiers we can see that the sequential model achieve a better Precision but worse Recall in contrast to the classification in the Packet level. On the other hand, the sequential model has the same Precision and better Recall than the classification in the Host level. In our case both methods seem to perform equally well. Although from theory we know that classification methods are hard to be implemented in botnet detection tasks.

In this task, classification was both trained and tested in the same scenario. For more generalized model we should have tried to train the model with one scenario and test with another one, as we did with the sequential model.

We can conclude that for our case, training and testing with Scenario 10, both classification and sequential model perform very well. The sequential model identifies all the infected host, but also mistakenly labels 1 benign as infected. On the other hand the classification makes some mistakes regarding the identified botnet flows. Nonetheless the precision of the classification method seem to be higher than the precision of the sequential model.

Code

The codes for this assignment can be found in `https://github.com/Stavroulakis/CyberDataAnalytics_CS4035/tree/master/Assignment_3`

References

[1] Pellegrino, Gaetano & Lin, Qin & Hammerschmidt, Christian & Verwer, Sicco. (2017). Learning Behavioral Fingerprints From Netflows Using Timed Automata. 10.23919/INM.2017.7987293.

[2] Abou-Assaleh, Tony & Cercone, Nick & Keselj, Vlado & Sweidan, Ray. (2004). Detection of New Malicious Code Using N-grams Signatures. 193-196.

[3] S. García, M. Grill, J. Stiborek, and A. Zunino. 2014. An empirical comparison of botnet detection methods. Comput. Secur. 45 (September 2014), 100-123. DOI=http://dx.doi.org/10.1016/j.cose.2014.05.011