Digital Forensics - Project Report

Project 1 - Packet classification for mobile applications

Eugen Saraci - 1171697
Università degli Studi di Padova

eugen.saraci@studenti.unipd.it

January 22, 2019

The large expansion of SSL/TLS in the last years has made it harder for attackers to collect clear text information through packet sniffing or, more in general, through network traffic analysis. The main reason for this is that SSL/TLS encrypts the traffic between two endpoints, which means that even though packets can still be easily captured, no useful information can be inferred from the packet's content without having the encryption keys.[1]

The authors of [1] and [2] showed that by training a machine learning algorithm with encrypted traffic data, one could correctly classify which actions a user performed while using some of the most common Android applications such as Facebook, Gmail, or Twitter. This could easily lead, through correlation attacks, to the full deanonimization of fake, privacy preserving identities.

In this work I try to reproduce the results achieved in [1] and [2] by implementing the classification model described in the papers.

1 Introduction and Notation

1.1 Actions and Flows

In order to easily understand this work, in the next two subsections I introduce two general concept used throughout this report.

[1] It is worth mentioning that the endpoints of the communication (i.e. source and destination IP addresses) are transmitted in clear text for routing purposes; by performing a DNS lookup of the addresses and attacker could easily infer what site a user is visiting.

1.1.1 Action and Action Label

An *action* is simply the action performed by a user while using one of the aforementioned Android apps. Examples of actions are: clicking on a profile page, tweeting a message, sending an email etc. Please note that "clicking on a profile page" is what I refer to as the *action label*, in many cases I use the words "action" or "action flow" to refer to the set of flows that represent that action. TODO: spiegare meglio sta roba

1.1.2 Flows

When a user performs an action some encrypted packets are exchanged with the destination server. A flow consists of the sequence of the byte sizes of the exchanged packets. If the packet is going from the user's phone to the server it is said to be *outgoing*; if the packet is coming from the server to the user's phone it is said to be *incoming* and it is marked with a "-" sign before the integer number representing its size.[2] An example of a 5 packet flow is: `[-12, 80, 90, -111, 30]`. Please note that a single action performed by the user usually generates multiple flows of different dimensions, by that follows that an action actually consists of multiple flows. The techniques used by the authors to determine which flows belong to which action, the ordering of the packets, the packet capturing system, the packet filtering system, and the statistical analysis on the flows will not be treated in this report since the starting point for this work comes when the dataset is already constructed.

1.2 Notation

- A: an action; it represents a sequence of flows;
- a: action label;
- F: a flow; it represents a sequence of packets;
- p: a single packet, it is an integer number representing the size in bytes of that packet.

Please note that all of the above can be subscripted by indexes; a subscripted element means that that element is the i-th element of a sequence, e.g. F_i is the i-th flow of a sequence of flows (possibly an action A).

By this follows that $A_i = [F_1, \ldots, F_n]_i$ and $F_i = [p_1, \ldots, p_m]_i$. Note that n and m are possibly (and probably) different for each flow F_i and for each action A_i, even for two actions A_i, A_j where $a_i = a_j$.

1.3 Dataset

The dataset consists of 252,151 rows (samples) and 12 columns (features), moreover, the data collected contains packets of different actions for 7 different Android applica-

[2] The "-" sign is just notation, packets cannot have a negative size.

action_start	**app**	**action_label**	**...**	**flow**
1383129102.11	facebook	open facebook	...	[-15, 75, 144]
1383129102.11	facebook	open facebook	...	[-55, -255, -333, 122, -55]
...	...	...	...	...
1383129102.11	facebook	open facebook	...	[12, 12, 155, 155, -18, 255]
1383129244.01	facebook	click menu	...	[78, -206]
...	...	...	...	...

Table 1: Some rows of the dataset.

tions: Facebook, Twitter, Gmail, Google Plus, Tumblr[3], Dropbox, and Evernote.

In Table **??** we can see the format of the dataset. I have purposively hidden some of the columns since I do not use them in this work. In the table we can see that each row contains the features of a single flow; to decide which flows belong to which action we compare the *sequence_start* field which is a fake but consistent timestamp of when the user started that action; When two rows have the same timestamp we can safely assume that they belong to the same action.

2 Machine Learning

Goal: as in Figure 1 we want a machine learning algorithm $\mathcal{L}$ that predict the *action label* a_i given the *action's flows* $A_i = [F_1, \ldots, F_n]_i$.

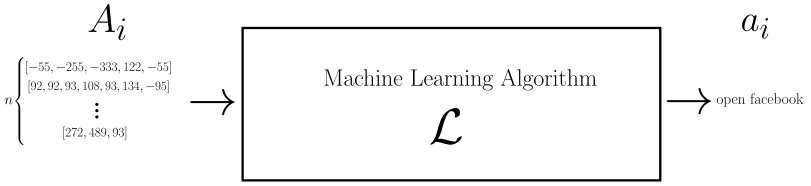

Figure 1: The ideal model predicting the label a_i for the a new input instance A_i.

Given the fact that each action generates multiple flows of different lengths, we know that standard supervised learning approaches are hard to apply. To see why standard approaches would not work we need to think about the structure of the input and output spaces. Our output space i.e. what we want to predict would be the *action label* a_i, while our input space, i.e. the predictors, would be the flows generated by a_i which we denote as $[F_1, \ldots, F_n]_i$. One way we could represent flows as features would be to have a feature for each flow F_j generated by the action a_i, and the value of the j-th feature would be the sequence of byte sizes of the flow $[p_1, \ldots, p_m]_j$. Because of the different number of flows for each action n we would immediately see that each row

[3]Because of some inconsistencies between the papers and the dataset, I could not provide a classifier for the Tumblr's actions.

could possibly have a different number of features. The main problem of this approach, other than the just mentioned diverse dimensionality, is that we are artificially defining features with no real justification; in other words, we have no reason to associate the first flow of an action A_i with the first flow of another action A_j by viewing them as the first feature of their respective samples.

To overcome this obstacle the original authors applied a two stage process. In the first stage they address their missing knowledge about the number of flows and their features by using an unsupervised clustering algorithm. The objective is to identify some kind of intermediate representation of the data $\Phi(A_i)$ that can be exploited by the second stage of the process, which will perform a canonic classification (Figure 2). In the remainder of this section we describe the algorithms used in the two stage process while my personal implementation is presented in section 3.

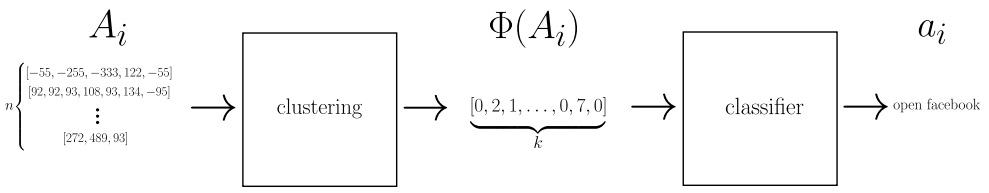

Figure 2: The two step classification of a new sample; first the new sample A_i goes through the clustering algorithm which computes the intermediate representation $\Phi(A_i)$, then $\Phi(A_i)$ is used by the classifier to predict the correct label a_i. k is the number of clusters, a more detailed description of k and Φ can be found in subsections 2.1.1 and 2.1.3.

2.1 Clustering

In the typical unsupervised clustering scenario the goal of the algorithm is to find k groups called clusters, where the *inter-cluster* similarity is very low while the *intra-cluster* similarity is maximized, meaning that samples belonging to one clusters are very similar to each other while still being very different from samples belonging to other clusters. Notice that the number of clusters k and the similarity function between samples have to be explicitly defined by the teacher. The input of the clustering algorithm is usually the whole dataset while its output is a list of integers $[c_i, \ldots, c_N]$ where N is the total number of samples and $c_i \in [1; k]$ is a an integer number that denotes the cluster to which the i-th sample has been assigned by the algorithm.

2.1.1 Hierarchical Agglomerative Clustering

In this specific instance I used (as suggested in the papers) a **Hierachical Agglomerative Clustering**, **HAC** for short. HAC starts by creating a cluster for each flow; at each step HAC reduces the number of clusters by merging (agglomerative) the closest clusters together based on the distance function and other parameters; HAC will stop

whenever the desired number of clusters k is reached. The structure that HAC exploits can be seen as a tree (hierachical), that structure is called dendrogram; in Figure 3 we can see the dendrogram for 20 samples (i.e. 20 flows). In my implementation the input consists in all the flows contained in the dataset[4], the number of clusters k is set to 250, and the distance function used to compute similarity between flows is the Dynamic Time Warping.

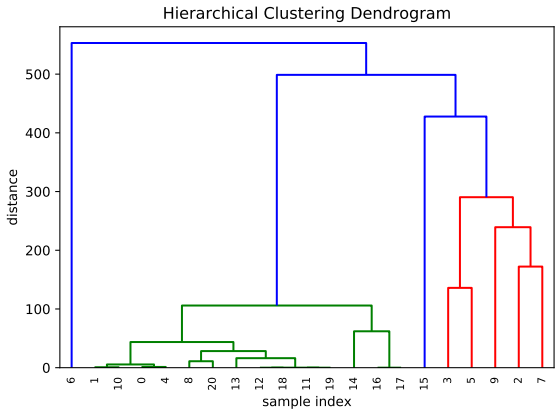

Figure 3: Dendrogram for 20 flows; the number of clusters can be seen by counting how many vertical lines get crossed by an imaginary horizontal line. Different horizontal lines will of course determine different clusters and different distance measures; the dendrogram allows the designer to decide at what level to draw the line i.e. which value to assign to k by looking at the distance value and at the hierarchy itself.

2.1.2 Dynamic Time Warping

Dynamic Time Warping or **DTW** is an algorithm generally used to find the best match between two time series even when they have different lengths and/or have repetitions or deletions in them. If we view every single flow F_i as a series of packets $[p_1, \dots p_m]_i$, we can use DTW to measure how similar two flows are. This similarity measure is needed by the HAC algorithm to compute the distances between clusters.

2.1.3 Feature Extraction

The HAC output is, as stated before, a list of N integers $[c_1, \dots, c_N]$ where c_i is the cluster assigned to the i-th input sample. This is different from what is reported in Figure 1, in fact the clustering result needs to be rearranged before being given as input to the classifier. The aim of the rearrangement is to actually generate a new intermediate dataset where the target we want to predict are still the actions' labels

[4]We are just using flows now, the concept of "actions" is being ignored for the moment since it is not really needed for the clustering stage.

action_label	**C1**	**C2**	**...**	**C250**
open facebook	0	11	...	6
writing search	1	1	...	5
back to news	0	0	...	0
open facebook	0	9	...	5
...	...		...	...

Table 2: Format of the new dataset. We can see that 11 flows of the first action ended up in cluster number 2, while 6 of them ended up in the last cluster.

but the features are now different. We generate k features for each action A_i; the value of the j-th feature is the number of flows of A_i that ended in the j-th cluster during the clustering algorithm. This whole process composed by the *clustering* and the final rearrangement is what we call feature extraction and denote with the symbol Φ, so $\Phi(A_i)$ is the intermediate representation of A_i and is considered to be a single input sample of the new intermediate dataset.

2.2 Classification

Now that we have a suitable dataset we can apply a supervised classification machine learning algorithm. In this scenario we split the data in two disjunct sets: *training* and *test set*; the first will be used to train the classifier while the other will test if the algorithm actually learned the concept that correlates the intermediate representation $\Phi(A_i)$ to the label a_i. The machine learning algorithm chosen is **Random Forest**.

2.2.1 Random Forest Classifier

The Random Forest Classifier is, as the name says, a classifier; its two main features can also be explained by analyzing its name:

- **Forest**: it is a collection of decision trees (which are (*weak*) classifiers themselves); if we want to classify a new instance on an already trained forest, what happens under the hood is that the forest will pass that instance to all of its trees and each one of them will return a prediction for that instance; the output of the forest will be the prediction voted the most by its trees.[5]
- **Random**: in Random Forest there are two degrees of randomness; during training phase each tree has to be build using:
  - a random subset of the total input data;
  - a random subset of features extracted from all the features available (usually $\sqrt{\text{total number of features}}$).

To better understand how Random Forest works it is worth mentioning the basic structuture and mechanisms we find in Decision Trees.

[5]Some variants will weight the vote of each tree differently based on their performance.

Structure of Decision Trees

- **nodes**: each internal node (root node included) can be represented by the subset of the samples it is working with (the root node is working with the whole dataset) and by the decision that has to be made at that node; the decision is just a boolean expression that gets applied to the all elements of the current subset.
- **arcs**: the arcs are the possible outcomes of the decision at a given node; the elements of the subset of the node will be sent down the arc (and the following sub-tree) they "satisfy".
- **leaf nodes**: leaf nodes are predictions (this means they are possible values of a or *action labels*), a leaf node is formed when all the elements of a given node have the same class/target/a/*action label*.

Mechanisms of Decision Trees The aim of the Decision Tree is to generate at each node the purest possible subsets, to do so it uses (im)purity measures (Information Gain) or dispersion measures (Gini Index). At each node only one feature is tested, the outgoing arcs of a node are the number of possible values that the feature can assume (if discrete) or a threshold value (if real), in which case only two arcs will be generated. Both the feature and the threshold (if any) are chosen based on the criteria cited previously. Adding to what have been said in the previous paragraph: a leaf node can be formed also when there are no more features to test (they have all been tested in previous nodes), in this case the prediction of the leaf node will be the *action label* with higher frequency on that subset. A part of a Decision Tree can be seen in Figure 4, the colors and their intensity are used to highlight which *action label* is more frequent at each node.

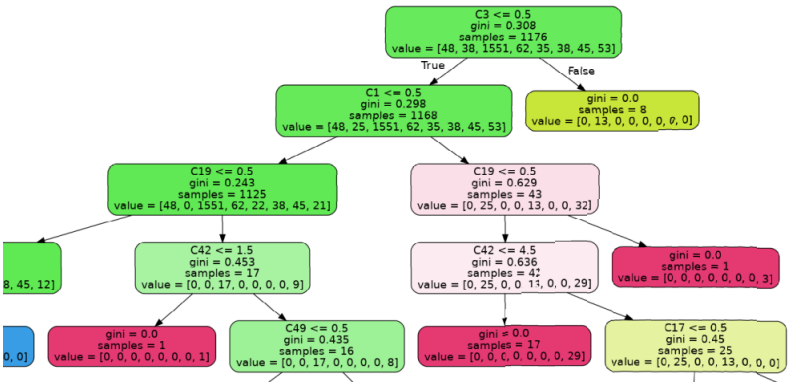

Figure 4: Small part of a single tree of the forest. At each node the value of a given feature is tested, the arcs opening to the left satisfy the condition while those to the right do not.

3 Evaluation

3.1 Experimental Setup

For this work I implemented two scripts using Python 3.6.5. The two scripts represent the two stages of clustering and classification shown in Figure 2; the first script, named `clustering.py`, performs the clustering and saves the intermediate representation as a .csv file called `[appname]_dataset.csv`. This file is then read as input by `classifier.py` which will consequently perform the classification using the Random Forest algorithm presented in Section 2.2.1. The computed performance measures will be printed to `stdout`.

It has to be noted that I trained a model for each different application, this means that the model that classifies Facebook actions has no way to classify Twitter actions since none of the Twitter flows has been shown to it during the clustering process or during the Random Forest training phase.

Most of the actions collected in the initial dataset are deemed to be not privacy-violating by the original authors; I do share their opinion, for this reason, before training the Random Forest, I renamed all the labels of the non relevant actions as *other*. This operation is done in memory by the `classifier.py` script, in this way the intermediate dataset stays untouched and many different configuration of relevant actions can be tried. The list of relevant action for each app can be found inside `classification.py`.

3.1.1 Computational Issues

Because of some issues with the functions provided by the *SciPy* library, I had to keep in memory the distance matrix of the flows. In the distance matrix D each entry (i, j) contains the value of $dtw(i, j)$ which makes D symmetric; this allows me to reduce by half the memory needed for the matrix since I could just compute the triangular upper (or lower) part of D. Unfortunately this "optimization" was not enough; the amount of memory needed was way higher that what my Google Cloud machine had. For this reason I decided to reduce the dataset to about 10,000 flows per application; again this is done in memory by the *clustering.py* script, therefore the original dataset is not modified. The number of actions in the initial and intermediate datasets can be seen in Table 3.

3.1.2 Differences from the original work

Apart from the smaller number of flows used for training phase, there are still other differences from the original work throughout my implementation. First of all the original authors weighted differently incoming and outgoing packets, secondly, they never considered all the packets of a flow, but, through statistical analysis, they decided take into consideration only packets who were to be found in specific hard coded ranges. Both these approaches allowed them to test different configurations which then lead to a maximization of the performance of the classifier. Same thing goes for the number of clusters: they evaluated the fitness of the clustering based on the random forest

	Original Dataset		Intermediate Dataset
	Flows	**Actions [unique labels]**	**Actions**
Dropbox	48462	15103 [33]	3138
Evernote	17014	6214 [32]	4239
Facebook	50319	12468 [41]	2494
GMail	9924	5644 [38]	5644
GPlus	33471	17573 [38]	5727
Twitter	36259	7334 [25]	2009

Table 3: The table shows the different number of actions for the different datasets. The smaller number of actions in the intermediate datasets is the result of limiting the number of flows to about 10,000 per application.

performances on it, this means that the number of clusters k they chose was obtained through this form of parameter tuning. While I do agree on the methodology they used, I am not able to follow the same path due mainly to time and computational constraints. Luckily, however, by looking at the images of the original work (Figure 5) I could grasp that the best value for k, the number of clusters, was close to 250, which is the value I used for this work.

3.2 Experimental Results

In Table 4 are shown the results using the same metrics employed in the original work: *precision* (P), *recall* (R), and *F1 measure* (F_1). Being this a multiclass problem (the multiple classes being the *action labels*), the shown results are the weighted averages of the measures obtained for each class. The scores are almost identical to those achieved in the original work as shown in Figure 5.

	P	R	F_1
Dropbox	0.87	0.91	0.88
Evernote	0.95	0.95	0.95
Facebook	0.99	0.99	0.98
GMail	0.86	0.91	0.88
GPlus	0.78	0.81	0.79
Twitter	0.99	0.99	0.99

Table 4: Precision, recall, and F1 values for the different apps.

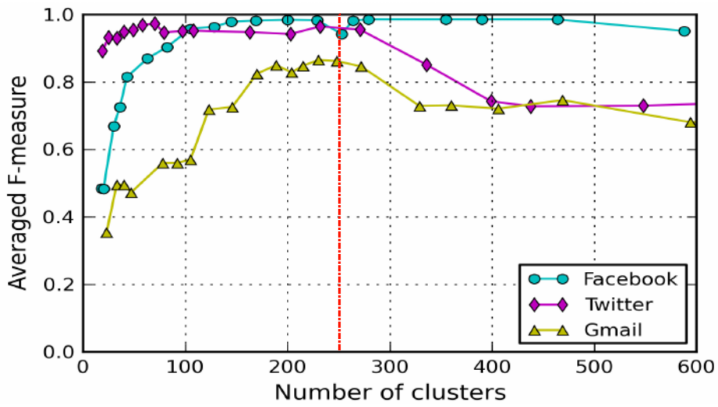

Figure 5: The performances of the classifier used by the original authors for different number of clusters. This figure is taken from [1] just for comparison; it shows that for $k = 250$ (marked by the red vertical line), the original authors and I achieved more or less same performances.

4 Final Notes

In conclusion, the aim of this work was to replicate what has been done in [1] and [2]; due to time and computational limitations I had to simplify my model, and luckily no loss of performance was encountered, allowing me to successfully reach the original results. Since the original papers might be a little outdated (the traffic data was captured in 2014), it might be interesting to perform the same kind of attack using current day's network traffic and evaluate if such issues still persist.

All the scripts, the datasets, the confusion matrixes, can be found at: https://github.com/esaraci/knocking.

References

[1] Mauro Conti, Luigi V. Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. 2015. Can't You Hear Me Knocking: Identification of User Actions on Android Apps via Traffic Analysis. In Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (CODASPY '15). ACM, New York, NY, USA, 297-304. DOI: https://doi.org/10.1145/2699026.2699119

[2] Conti, M., Mancini, L. V., Spolaor, R., & Verde, N. V. (2016). Analyzing android encrypted network traffic to identify user actions. IEEE Transactions on Information Forensics and Security, 11(1), 114-125.