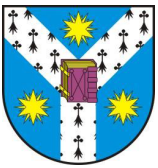

University Alexandru Ioan Cuza
Faculty of Informatics

Collection of Hybrid Models used for Cyber-Threats Detection

Author:
Cristina
VATAMANU

Supervisor:
Prof. Dr. Henri
LUCHIAN

0.1 History of Malware Evolution

The malware landscape has evolved over time in many shapes and forms, becoming each day more and more complex. The beginning of malware relies on practical jokes between individuals which had a passion for computer science. Starting with the rise of the Internet things took a financial turn in the cyber space and what was once a proof of concept, starting from that moment on, could produce money. Starting with 2005, 2006 different kinds of trojans were created, serving a wide number of purposes, but with the same result, money gain: stealing banking credentials, promoting ads, deceiving people through phishing and developing credit card scams and so on. Since the focus was to gain money, the threat-actors had to make sure that their files remained undetected as long as possible and, for that, many techniques were developed during this time. There were three main approaches:

1. change the binary file faster than the security solution can update (defensive techniques)
2. stay under the radar and make analysis and detection as hard as possible (protective techniques)

3. take actions at a system level in order to stay undetected (offensive techniques)

The huge number of malicious samples seen in the wild is an issue with which security solutions are still fighting in the present days. But starting with 2010 it was observed that financial gain was not the only purpose of the threat-actors. Over the last years, malicious threats were used as weapons for different purposes as stealing information, intelligence gathering, used by governments for national espionage or for population surveillance. It is clear that the cyber-threat landscape has evolved over time and it is in continuous change, becoming more sophisticated, complex and aggressive. In order to face all the attacks in the wild, many researches were done in this direction and security solutions started to integrate technologies based on machine learning models. Detection methods, based on machine learning algorithms must take into consideration numerous aspects, in order to be efficient and secure such as:

1. A model must be able to adapt fast in order to keep up with the continuous changing landscape.
2. Most of the technologies used for detection work in a blocking manner. It is important that the overhead brought in a

system (time needed to take a decision if a file is clean or not) is as minimum as possible.

3. The models integrated in a security product have to have a small memory footprint in order to avoid an overhead and to assure that the system would work properly.
4. One of the main concerns over time was to create aggressive proactive models able to detect and block new unknown threats, but in the same time to keep a false positive rate as low as possible.

This thesis is covering some hybrid mechanisms, meant to be used against large datasets of files, approaches that are trying to find the best equilibrium between all the four restrictions mentioned above.

0.2 Hybrid Models used for Cyber-Threats Detection

Malicious threats have a history of approximately 25 years and they've changed forms over the years, being very dynamic, covering a wide area of malicious techniques, hidden interests and caused damages. The objective of a security solution is to protect as well as possible the users against cyber-threats. Integrating a classification

model within a security product is a complicated process. Although, in theory, a model generates good results, there are four main, strict restrictions imposed by practice use:

1. *Detection Rate: TPR* - The main task of a security product is to protect and detect threats and that is directly translated as the detection rates of the different integrated technologies.
2. *False Positive Rate: FPR* - A security solution must protect, but it should not detect everything. Having a low number of misclassified legitimate files is mandatory.
3. *Performance Impact (or Evaluation Time): ET* - Performance impact is translated as the time needed by the model to perform all the necessary calculus in order to take a decision if a record belongs to one class or the other.
4. *Memory Footprint: MF* - Memory Footprint, for a particular algorithm, refers to the necessary amount of space to store the resulted model, usually illustrated by the feature IDs and the corresponding weighs set.

When analyzing a model, one must always measure these restrictions, because any modification on the algorithm parameters would affect one or more of these performance indicators.

0.3 Database used in practical experiments

DB-CAS-BDT

	Count
Clean Records	2082322
Malicious Records	242805
Total	2325127
Number of features per record	6275

Table 1: Dataset structure used in present research

0.4 Linear Classifier

Some of the present research is based on a modified linear classifier, called One Side Class Perceptron - OSCP([1]) and this section will give some details of its main properties. The classifier modifies Rosenblatt's Perceptron algorithm ([2]) in order to satisfy some practical restrictions of the cyber-security industry.

The OSCP algorithm main focus is to correctly classify all the records from a certain class. It was developed to minimize the number of false positives. The way it succeeds to do that is by adding an additional step at the end of each training iteration, a step which modifies the hyper-plane position in such a way as to guarantee that

all the elements from a certain class are correctly classified. All the training iterations, that use all the records, are followed by such a step, entitled *"Class Minimize Step"*, that will continue to train the model using only the elements of that particular class. At the end of each iteration, on one side of the hyper-plane there will be *all* the elements from the first class, and some from the second class, and on the other side of the hyper-plane there will be *only* elements from the second class.

For a simple illustration a two-dimensional space example of a training iteration is illustrated in Figure 1. The first figure (a) represents the dataset of records D. After the training step, which uses all the records, the position of the hyper-plane will look like in figure (b). Before proceeding to the next training iteration the class minimize step is applied, where the model will be training only with the records belonging to one class S, the benign records in this example (the blue squares). After this step, the hyper-plane will change in such a way that all the clean records will be correctly classified (figure (c)).

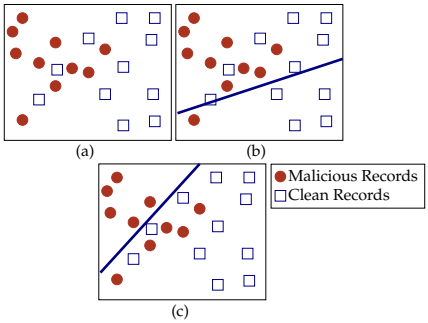

Figure 1: One Side Class Perceptron - OSCP training iteration. Training Step (b). Class Minimize Step (c)

There are several advantages of this algorithm in terms of practical usage:

- It has zero false positives during the training process
- It has a very low number of false positives on testing
- It is fast in term of training time and testing time
- Because of the way it was developed it can be parallelized, so the training process can be distributed on several machines
- It uses a small amount of memory

This algorithm was first developed as a filter for minimizing the number of false positives. The way it works, it also has some disadvantages:

- Low detection rate (lower than the one obtained by Rosenblatt Perceptron)
- Results are very bad if elements belonging to different classes have the same set of features, this is why it is very important to filter the noise from the dataset.

0.5 Cascading Learning

Having the OSCP algorithm optimize to have good results on three of the indicators (false positive rate, memory footprint and performance impact), the next step was to see how the detection rate can be improved. The first approach considered was cascading learning. Cascading is a particular class of ensemble learning, which combines different machine learning models. As in ensemble theory, cascading uses the information provided by the output of a classifier as additional data for the following classifier (Adapting Boosting). The present part of the research is based on a multi-stage classifier. At each stage one model, which is able to isolate a subset if records as correctly classified, will remove that subset from the database. This way, the next classifier will focus only on the data which is more difficult to classify. Algorithm 1 describes the approach considered.

Algorithm 1 Cascading Models

1: $D \leftarrow \bigcup_{i=1}^{n} x_i$ - the database
2: $M \leftarrow \{M_1, M_2, ...M_n\}$ - set of models
3: **function** $Cascade(D, M)$
4: **while** $|D| > 0$ **do**
5: Choose M_i from M, taking into consideration D
6: $C \leftarrow M_i(D), 0 < |C| < |D|$
7: $D \leftarrow D \setminus C$
8: **end while**
9: **end function**

The idea behind this approach is that different feature sets describe better a particular area of the dataset than other feature sets. This means that one can build different models, that use certain set of features in order to best describe a part of the dataset, or in other words, have specialized models for certain parts of the database. Since at each cascading stage, a classifier must isolate a subset of records, which should be eliminated from the database of the following stages, the present research took advantage of the OSCP classifier property. The algorithm assures that on one side of the hyper-plane are all the elements from one class and some misclassified records from the second class and on the other side are records correctly classified only belonging to the second class, these last records can be eliminated from the database of the next cascading stage. In other words, at

each cascading stage, an OSCP algorithm is trained obtaining at the end of the stage a model and the dataset for the next stage. The algorithm stops either when all the records are correctly classified or when the maximum number of stages is reached (condition imposed in practice). During this study, various combinations of OSCP algorithms were used across the cascading stages:

- Cascading OSCP one class
- Cascading OSCP alternative classes
- Cascading OSCP k-alternative classes
- Cascading OSCP best class
- Cascading OSCP both classes

The best results were obtain with the last solution which will be further used. An example of a two-dimensional space data on which an OSCP Both Classes is applied is illustrated in Figure 2.

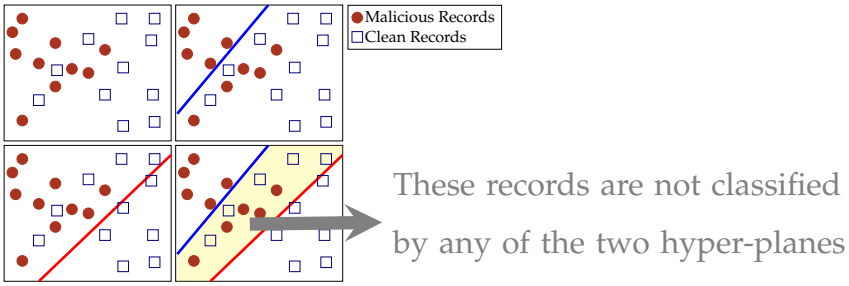

Figure 2: OSCP Both Classes

As it can be observed, the algorithm isolates two subsets of correctly classified samples (one belonging to the class of clean files and one to the class of malicious files) and a set of records which are not classified by any of the two hyper-planes. An interesting aspect to be considered in cascading learning is represented by the features. Having a large amount of features describing the data to be classified imposes the necessity of a feature selection process. Usually, the feature selection is performed only once, right before training a classifier. However, when it comes to cascading learning, one can choose to perform this process at the beginning of each stage, selecting a different set of features each time. This way, every model will be trained using features that are more appropriate for the current subset. Another observation during this study was that, in all performed experiments, the algorithm exits with the stop condition imposed by practice, the maximum number of stages was always reached. To identify the appropriate maximum limit, multiple tests were conducted, specifically the tests tried to execute the algorithm with as many stages as possible in order to correctly classify all the

data. Since the dataset is very big, subsets from it were selected. The first dataset considered had 33240 records (3194 malicious files and 30046 clean ones). The algorithm using an OSCP Both Classes classifier, trained for 500 iterations managed to correctly classify all the data in 19 cascading stages.The next step was to increase 10 times the size of the dataset. The next test database contained 332404 records: 31940 malicious files and 300464 clean ones. At the 39th stage, the current model was not able to isolate any record to be discarded. From that particular point, the number of training iterations doubled, and for the next stages the OSCP classifier was trained for 1000 iterations. Unfortunately, the algorithm was blocked again at the 46th stage and again at the 58th stage. During the last 5 cascading stages only one record per stage was discarded, and at the end of the 58th stage 113560 records still remained unclassified. The conclusion of the experiment was that the algorithm has and will continue to have a very slow decrease. Only the first three stages managed to correctly classify and discard from the dataset more than half of the records. From the 6th stage the decrease started to be and remained almost

imperceptible with the exception of the 39th and 46th stages, which correspond to the two cascading stages for which the number of training iterations for the OSCP classifier was doubled. After these tests were conducted the maximum limit for the cascading algorithm was 10 stages, and it was considered to be a reasonable choice. During the process of data analysis, the OSCP-Both classes training was also evaluated. It was observed that, in some cases, the data has an odd behavior. Starting with the second cascading stage, during OSCP training, between 50% and 70% of the iterations produced this action: the number of correctly classified records alternated between a small value and a high value and only during the last 50%-30% of the iterations, the detection rate seemed to have a linear increase, but the OSCP detection rate did not managed to reach a threshold. In order to observe its performance, an experiment was conducted and, starting with the second cascading stage, the OSCP-Both classes algorithm was trained for 3000 iterations. The experiment is illustrated in Figure 3.

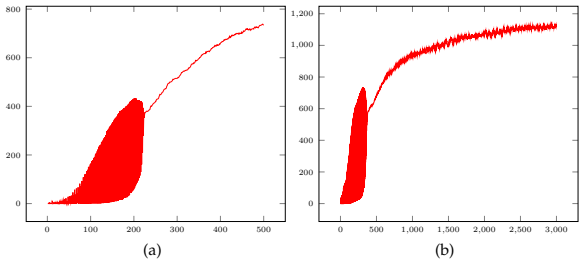

Figure 3: Training process for OSCP-Both Classes with 500 iterations (a) and 3000 iterations (b) during the second cascading stage

As it can be observed, the OSCP-Both Class classifier reached a threshold in terms of detection starting with the iteration 2000. Since only 10% of the iterations presented the previously described behavior (detection rate alternations), the detection boost was more visible.

The experiment highlighted that, better results are tightly related to the number of iterations that the classifier is trained for. But in practice, being able to train a classifier in a decent amount of time, usually comes with setting restrictions on its parameters and the number of iterations is one of them. Given the fact that cascading learning approach considered in this study, discards at each stage a number of records, the datasets for the next stages will get smaller and smaller, which means the processing time will decrease as well. One can take

advantage of this property and can trade the gain in processing time with improving the next classifiers.

The first technique applied in this part of the study was to increase, progressively, the number of OSCP training iterations with each cascading stage. On the first stage the classifier was trained for a constant number of iterations, imposed by practical reasons. Starting with the second stage, when almost half of the records are discarded, the number of training iterations was increase at each stage with a factor. Along with the number of training iterations, another parameter of great importance is represented by the set of features. Since the model is meant to be used in practice, this, usually means that it is needed to perform feature selection. In a similar manner as the previous described technique, an adjustment can be performed regarding the number of features. On the first stage the classifier was trained with a constant number of features, imposed by practical reasons. Starting with the second stage, this number was increase at each stage with a factor. A combination of the two techniques was also considered, where both the

number of training iterations and the number of features used in training for the OSCP-Both Class classifier were adjusted at each cascading stage. The previously described studies were tested against the database DB-CAT-BDT. The number of training iterations and the number features used in training, imposed by practical reasons were both set to 500. The 500 most important features were selected using the feature selection algorithm F-score [3].All the algorithms were tested by performing a 3-fold cross validation against the dataset. Four new cascading models were created, the factor w taking sequentially 4 different values. Particularly:

$$numberOfFeatures = 500 + stageNumber \times w,$$

$$where: w \in \{100, 200, 300, 400\}, stageNumber = \overline{0,9}$$

Second, the same technique was applied for the number of training iterations of OSCP-BC classifier, obtaining two new models. Particularly:

$$numberOfIterations = 500 + stageNumber \times k,$$

$$where: \ k \in \{100, \ 200\}, stageNumber = \overline{0,9}$$

The last experiment refers to adjusting both the feature number and the number of training iterations. One new model was created, CAS-200F-200IT, both parameters being adjusted with a factor equal to 200. All the results are summarized in Table 2:

Algorithm	FP Rate	Detection Rate	Training Time
OSCP-BC	0.0069 %	29.7832 %	2:05:13
CAS	0.0159 %	34.9699 %	3:09:24
CAS-100F	0.0334 %	45.9307 %	5:07:26
CAS-200F	0.0374 %	49.9215 %	6:07:44
CAS-300F	0.0342 %	49.4652 %	7:00:54
CAS-400F	0.0299 %	48.5496 %	7:51:11
CAS-100IT	0.0220 %	36.1729 %	6:49:06
CAS-200IT	0.0242 %	36.6923 %	9:16:27
CAS-200F-200IT	0.0537 %	56.8767 %	13:39:01

Table 2: Results obtained in practical experiments for cascading algorithms

0.6 Collection of hybrid ensembles based on Binary Decision Tree

Decision tree learning represents a model which translates observations about certain elements to conclusions

about that item value or label. A Decision Tree consists of a root node, several decision nodes and terminal nodes (the leafs).

0.6.1 Binary Decision Tree Split

In the present study the root node holds all the records from the database DB-CAS-BDT. At each decision node, a condition on a certain feature is evaluated. These conditions are used to split the current dataset into two subgroups. If a record has a specific feature set on true, then that record is saved in the right-node (sub-cluster), otherwise it will be saved in the left-node. The set of features used in successive evaluations are chosen using various scores which represent an important element in the splitting process. At each decision node, the feature with the highest score is chosen for evaluation. The function score used in this research is named Median Close (MC) and it is described by the Equation 1.

Considering:
$D \leftarrow \bigcup_{i=1}^{n} \left\{ x_i | x_i \in \mathbb{R}^m \right\} - the\ set\ of\ records$
$F \leftarrow \bigcup_{j=1}^{m} F_j - the\ features\ set$
$\forall F_j, j = \overline{1, m}$

$$MC_{F_j} = \frac{(1 - |\frac{countClean}{totalClean} - 0.5|) + (1 - |\frac{countMalicious}{totalMalicious} - 0.5|)}{2} \quad (1)$$

where the following notations are used:

- $countClean$ - the number of records belonging to the clean class for which the given feature is set to true
- $totalClean$ - the total number of records from the clean class
- $countMalicious$ - the number of records from the malicious class for which the given feature is set to true
- $totalMalicious$ - the total number of records from the malicious class

0.6.2 Hybrid models using Binary Decision Tree split and the OSCP classifier

First, the binary decision tree was combined with the OSCP-Both Classes classifier (Figure 4). On each cluster an OSCP-BC algorithm was trained for 500 iterations, using 500 features, selected with F-score function.

The detection rate and the false positive rate, training time and memory footprint was computed for 11 BDT depths (*Depth 0* illustrating the results obtained using the entire dataset DB_CAT_BDT), and it is exemplified in Figure 5

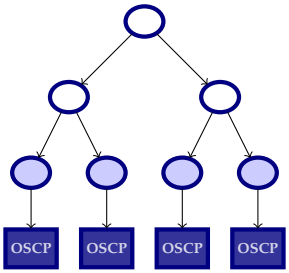

Figure 4: Binary Decision Trees combined with OSCP Algorithms

As it can be seen in Figure 5, the BDT Split technique clearly improves the detection rate, but there are several observations that have to be made regarding this experiment:

- The detection rate increases from 29.78% on the initial database to 92.62% when 1024 clusters are used, giving an approximately 62% improvement.
- False Positive rate increases 500 times, from 0.006% to 1.73%. False Positive rates over 1% are not recommended because it becomes very difficult to solve the misclassified legitimate files with an whitelisting process.
- The size of the model is also affected, and it grows exponentially with the number of BDT levels used.
- The performance is also impacted. It was measured in training time and it goes from almost 10 minutes to half an hour.

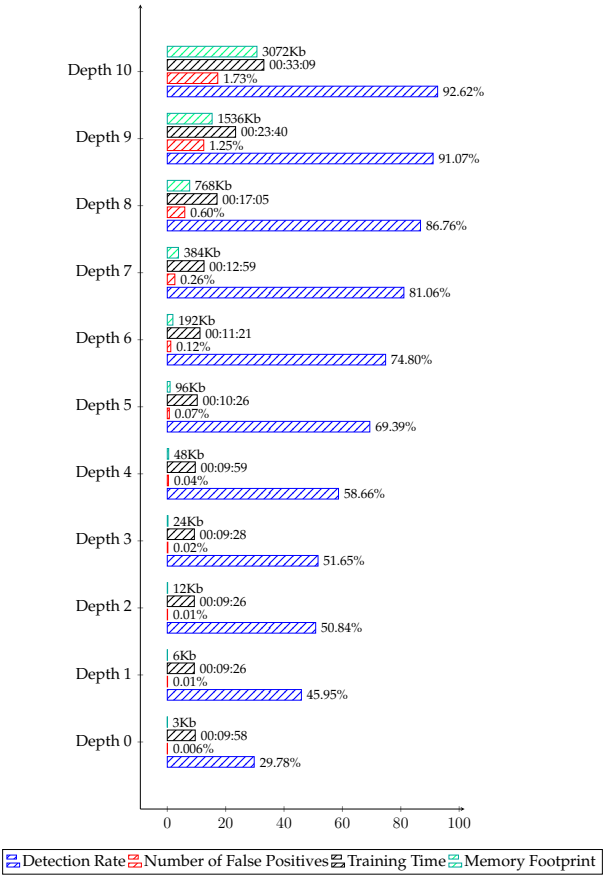

Figure 5: Detection Rate, False Positive Rate, Training Time and the Memory Footprint for OSCP classifier in correlation with different depths for the Binary Decision Tree Split

0.6.3 Hybrid models using Binary Decision Tree split and Support Vector Machine based classifiers

The next step in the present study was to evaluate the BDT Split method with different Support Vector Machine

based classifiers. Since this was a comparative study, a python library called *sklearn* was used to implement the SVM models [4]. Two types of kernel functions were considered: linear and Radial Basis Function. Each model was trained and tested (3 fold-cross validation) on the final clusters obtained through the BDT split (Figure 6).

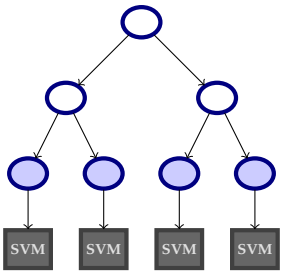

Figure 6: Binary Decision Trees combined with SVM Algorithms

Since it is very difficult to train SVM-based models on large datasets the datasets obtained with the BDT splits using depths 0, 1 and 2 were not used in this experiment. Aside for the kernel function, the algorithms were provided a parameter C, which accordantly to the authors *"trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly"*. By default, this parameter is equal

to 1. Three such values were tested: 0.1, 1 and 10. First the results obtained by the first three classifiers (linear kernel function) are illustrated in Figure 7.

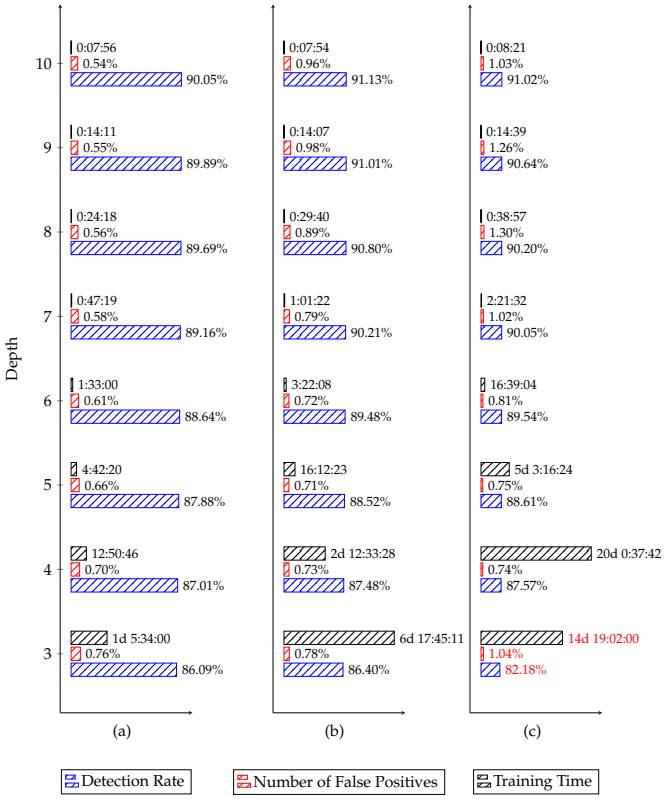

Figure 7: Detection Rate, False Positive Rate, Training Time for SVM classifier using a Linear Kernel function with the parameter C set to 0.1 (a), 1.0 (b), 10.0 (c) in correlation with different depths for the Binary Decision Tree Split

Some interesting facts were observed analyzing the results:

- As expected, it took a big amount of time to train the SVM models with linear kernel function on big clusters. Even more, the larger the parameter C is, the more time is needed to train the model. Training the models on the dataset obtained by the BDT split with three levels was not feasible, the training process taking more than a day. The third model ($C = 10$) was forcedly terminated after 14 days and 19 hours because after this amount of time only two models were generated and tested. This is way, the corresponding results are highlighted in red.
- Although it exists an increase in term of detection rate correlated with the number of clusters, it is not as sizable as the one observed when OSCP classifier was tested in the same manner.
- The parameter C does not seem to have a great influence on the detection rate, for all the three values (0.1, 1, 10) the detection rate stays between 86% and 91%.
- Not in all cases the models having the highest detection rate also have the highest false positive rate, as it could be seen in the experiments involving the OSCP classifier and the parameter C influenced the false positive rates, the greater the value of C is, the higher the false positive rates.

The same experiments were conducted on the other three models, using the RBF kernel function. The results obtained after training the classifier on the 8 databases are highlighted in Figure 8.

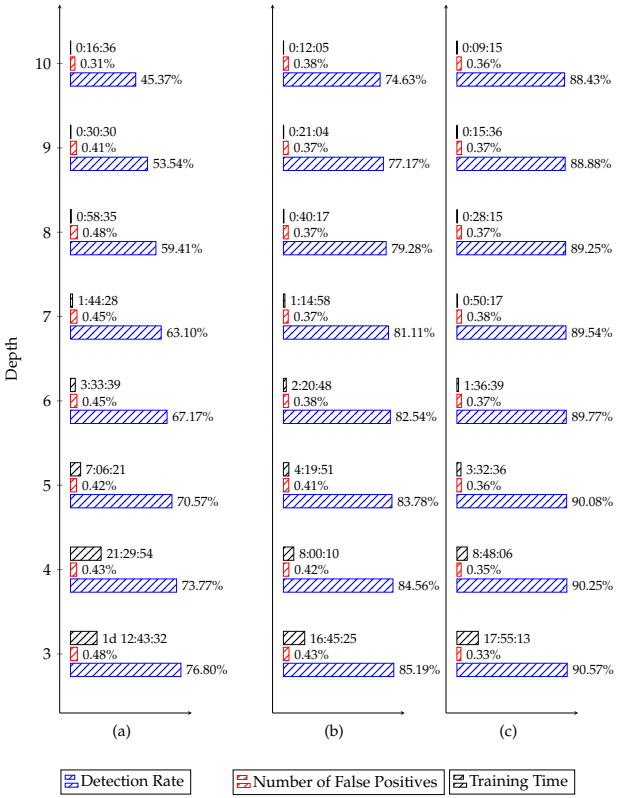

Figure 8: Detection Rate, False Positive Rate, Training Time for SVM classifier using a RBF Kernel function with the parameter C set to 0.1 (a), 1.0 (b), 10.0 (c) in correlation with different depths for the Binary Decision Tree Split

Several observations were drawn regarding these results:

- Although training time still depends on the size of the dataset, these models preformed much better than the ones using the linear kernel functions. The datasets built after the split done

through BDT with depths three and four still remain infeasible for practical use. The training processes done on smaller clusters (depths 8, 9, 10) have been completed under an hour.

- It seems that with the increase of the number of clusters, the detection rate decreases, in contrast with the experiments previously performed where the higher the depth of the binary decision tree, the higher the detection. Even more, this decrease is influences by the value of parameter C ($\approx 30\%$ when $C = 0.1$, $\approx 5\%$ when $C = 1$, $\approx 2\%$ when $C = 10$).
- If previously the parameter C did not seem to influence the detection rate of the models using a linear kernel function, here it can be easily seen that the model with the highest value for the parameter C has the highest rates in term of detection.
- The false positive rates obtained by these models are lower than the ones from the previous experiment, most of them under 0.5%. This time the parameter C does not seem the influence the number of misclassified clean files.

0.7 Hybrid models combining Binary Decision Tree split and Cascading learning

After studying these two methods for boosting detection, the next stage of the research was to combine the BDT split with different cascading approaches previously studied to see if the detection can be further improved.

0.7.1 Hybrid models using Binary Decision Tree split and 10 stages Cascading classifier

First, on each final cluster obtained through a BDT split, a cascading approach was applied as it can be seen in Figure 9.

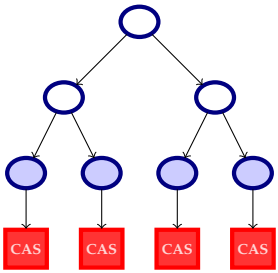

Figure 9: Binary Decision Trees combined with Cascading Algorithms

0.7.2 Hybrid models intercalating Binary Decision Tree split and cascading classifier

The present study analyzed another approach on how to combine the BDT split technique and the Cascading classifier. The idea started with the observation that the number of discarded records decreases at each cascading stage. More precisely, most of the samples were discarded during the first stage. This method tries to solve

this problem by intercalating the BDT split with the cascading stages and it is illustrated in Figure 10

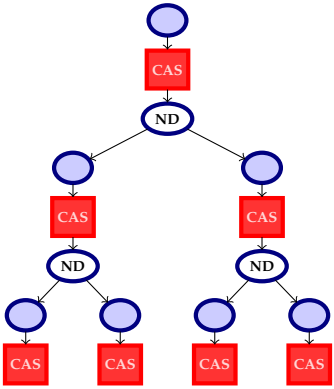

Figure 10: Intercalate cascading stages with splits performed by the Binary Decision Tree

The first step is applying a cascading stage on the initial dataset. The records correctly classified by the two models ($Model_{clean}$ and $Model_{malicious}$) are discarded, so the next operations will focus only on the records which the previous models could not take a decision. The second step is to apply the BDT split on the incorrectly classified subset returned by the previous algorithm. The leaf nodes (the final clusters) will then be sent as parameter to the previous algorithm. The results obtained are illustrated in Table 3

Algorithm	FP Rate	Detection Rate	Training Time
OSCP-BC	0.006 %	29.78 %	2:05:13
CAS	0.015 %	34.96 %	3:09:24
BDT3-CAS	0.109 %	59.94 %	4:16:12
BDT4-CAS	0.261 %	70.07 %	4:48:08
BDT4-CAS-3000	0.508 %	76.46 %	6:56:02
BDT4-CAS-200F-200IT	0.599 %	83.10 %	5:11:00
CAS-BDT1-200F-200IT	0.869 %	84.02 %	5:15:12

Table 3: Results obtained in practical experiments for cascading algorithms intercalated with BDT splits

0.8 Conclusions

The present study proposes different hybrid solutions meant to increase the detection rate, but also a detail analysis of practical restrictions which one must consider when creating a protection mechanism which is meant to be integrated in a security product. The first approach refers to cascading learning, where multiple classifiers are combined in such a way that one eliminates the correct classified records so that the next one could focus only on those elements which are difficult to classify. The second approach is an ensemble which combines a Binary Decision Tree split with different classifiers. A combination between these two methods was

also studied.

All the models are trained and tested on a dataset with more than two million of records. During the practical experiments four important performance indicators are analyzed: the detection rate, the false positive rate, the memory footprint and the evaluation time. Such indicators are very important, first to see if the approaches are proper for practical use, but also to be able to fit the solutions in certain categories. It is important to know which of the indicators have a strict limit in order to correctly identify the complementary technologies so that a proper protection can be provided.

0.9 Appendix: Summary overview regarding the performance indicators for all studied algorithms

The last section is meant to gather all the experimental results obtained during this study and to evaluate the performance indicators for all the algorithms present in this research: the detection rate, the false positive rate, the memory footprint and the performance impact (measured as evaluation/testing time).

Indicators

Terminology

- Positives - P: the number of malicious samples
- Negatives - N: the number of clean samples
- True Positives - TP: number of malicious samples identified as malicious by a classifier
- True Negatives - TN: number of clean samples identified as clean by a classifier
- False Positives - FP: number of clean samples identified as malicious by a classifier
- False Negatives - FN: number of malicious samples identified as clean by a classifier

Detection Rate

Detection rate or True Positive Rate is defined as the ration between the number of true positives and the total number of positives:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

False Positive Rate

False positive rate or fall-out is defined as the ration between the number of false positives and the total number of negatives:

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

Memory Footprint
Memory footprint is defined as the size of the model or the amount of storage needed to memorize a model. This indicator is dependent of the number of features used. Let m be the number of features used by the classifier C and N the number of support vectors in the case of Support Vector Machine classifiers. Each feature has an ID (2 bytes) and a corresponding value (4 bytes).
Taking this into considerations the following memory footprints can be defined:

- $MF_{OSCP} = 6 \times m$
- $MF_{OSCP-BC} = 2 \times (6 \times m) = 12 \times m$
- $MF_{CAS} = noStages \times 12 \times m$
- $MF_{CAS-kIT} = noStages \times (12 \times m)$
- $MF_{CAS-zF} = \sum_{i=0}^{noStages-1} (m + i \times z) \times 12$
- $MF_{CAS-zF-kIT} = \sum_{i=0}^{noStages-1} (m + i \times z) \times 12$
- $MF_{BDTx-OSCP} = 2^x \times (6 \times m)$
- $MF_{BDTx-SVM-linear} = 2^x \times (6 \times m)$
- $MF_{BDTx-SVM-rbf} = 2^x \times (N \times m \times 6 + N \times 6 + N \times 1)$
  $= 2^x \times N \times (6 \times m + 7)$
- $MF_{BDTx-CAS} = 2^x \times (noStages \times (12 \times m))$
- $MF_{BDTx-CAS-3000} = 2^x \times (noStages \times (12 \times m))$

- $MF_{BDTx-CAS-zF-kIT} = 2^x \times (\sum_{i=0}^{noStages-1} (m + i \times z) \times 12)$
- $MF_{CAS-BDTx-zF-kIT} = \sum_{j=0}^{x} 2^j \times (\sum_{i=0}^{noStages-1} (m + i \times z) \times 12)$

Performance Impact

Is defined in the present research by the evaluation time, meaning the amount of time needed to compute all the calculus necessary to take a decision about a record, if it belongs to the clean class or to the class of malicious files.

The time needed to perform an operation differs from one operation to another, but to simplify the calculus, in the present study it is consider:

$$T_{mul} = T_{add} = T_{compare} = T_{assign} = \Delta$$

The evaluation time for the SVM models will be computed relative to the average number of support vectors per cluster. The following evaluation time can be defined for each studied model:

- $ET_{OSCP} = m \times \Delta + 1$
- $ET_{OSCP-BC} = 2 \times (m \times \Delta + 1)$
- $ET_{CAS} = noStages \times (2 \times (m \times \Delta + 1))$
- $ET_{CAS-kIT} = noStages \times (2 \times (m \times \Delta + 1))$
- $ET_{CAS-zF} = \sum_{i=0}^{noStages-1} ((m + i \times z) \times \Delta + 1) \times 2$
- $ET_{CAS-zF-kIT} = \sum_{i=0}^{noStages-1} ((m + i \times z) \times \Delta + 1) \times 2$
- $ET_{BDTx-OSCP} = x \times \Delta + m \times \Delta + 1 = \Delta \times (m + x) + 1$
- $ET_{BDTx-SVM-linear} = x \times \Delta + m \times \Delta + 1 = \Delta \times (m + x) + 1$
- $ET_{BDTx-SVM-rbf} = x \times \Delta + N \times (2 \times \Delta + m \times \Delta + 3 \times \Delta) + \Delta$
  $= \Delta \times (N \times (m + 5) + x + 1)$
- $ET_{BDTx-CAS} = x \times \Delta + noStages \times (2 \times (m \times \Delta + 1))$

- $ET_{BDTx-CAS-3000} = x \times \Delta + noStages \times (2 \times (m \times \Delta + 1))$
- $ET_{BDTx-CAS-zF-kIT} = x \times \Delta + \sum_{i=0}^{noStages-1} ((m + i \times z) \times \Delta + 1) \times 2$
- $ET_{CAS-BDTx-zF-kIT} = x \times \Delta + x \times (\sum_{i=0}^{noStages-1} ((m + i \times z) \times \Delta + 1) \times 2$

Having all the indicators defined for each studied model, the Table 4 summarizes all the results:

Algorithm	TPR	FPR	MF (bytes)	ET(No. Operations)
OSCP	29.78 %	0.006 %	3000	$500 \times \Delta + 1$
OSCP-BC	29.78 %	0.006 %	6000	$1000 \times \Delta + 2$
CAS	34.96 %	0.015 %	60000	$10000 \times \Delta + 20$
CAS-100F	45.93 %	0.033 %	114000	$19000 \times \Delta + 20$
CAS-200F	49.92 %	0.037 %	168000	$28000 \times \Delta + 20$
CAS-300F	49.46 %	0.034 %	222000	$37000 \times \Delta + 20$
CAS-400F	48.54 %	0.029 %	276000	$46000 \times \Delta + 20$
CAS-100IT	36.17 %	0.022 %	60000	$10000 \times \Delta + 20$
CAS-200IT	36.69 %	0.024 %	60000	$10000 \times \Delta + 20$
CAS-200F-200IT	56.87 %	0.053 %	168000	$28000 \times \Delta + 20$
BDT1-OSCP	45.95 %	0.011 %	6000	$501 \times \Delta + 1$
BDT2-OSCP	50.84 %	0.013 %	12000	$502 \times \Delta + 1$
BDT3-OSCP	51.65 %	0.022 %	24000	$503 \times \Delta + 1$
BDT4-OSCP	58.66 %	0.045 %	48000	$504 \times \Delta + 1$
BDT5-OSCP	69.39 %	0.072 %	96000	$505 \times \Delta + 1$
BDT6-OSCP	74.80 %	0.125 %	192000	$506 \times \Delta + 1$
BDT7-OSCP	81.06 %	0.261 %	384000	$507 \times \Delta + 1$
BDT8-OSCP	86.76 %	0.603 %	768000	$508 \times \Delta + 1$
BDT9-OSCP	91.07 %	1.251 %	1536000	$509 \times \Delta + 1$
BDT10-OSCP	92.62 %	1.732 %	3072000	$510 \times \Delta + 1$
BDT3-SVM-linear-C0.1	86.09 %	0.76 %	24000	$503 \times \Delta + 1$
BDT4-SVM-linear-C0.1	87.01 %	0.70 %	48000	$504 \times \Delta + 1$
BDT5-SVM-linear-C0.1	87.88 %	0.66 %	96000	$505 \times \Delta + 1$
BDT6-SVM-linear-C0.1	88.64 %	0.61 %	192000	$506 \times \Delta + 1$
BDT7-SVM-linear-C0.1	89.16 %	0.58 %	384000	$507 \times \Delta + 1$

Algorithm	TPR	FPR	MF (bytes)	ET(No. Operations)
BDT8-SVM-linear-C0.1	89.69 %	0.56 %	768000	$508 \times \Delta + 1$
BDT9-SVM-linear-C0.1	89.89 %	0.55 %	1536000	$509 \times \Delta + 1$
BDT10-SVM-linear-C0.1	90.05 %	0.54 %	3072000	$510 \times \Delta + 1$
BDT3-SVM-linear-C0.1	86.40 %	0.78 %	24000	$503 \times \Delta + 1$
BDT4-SVM-linear-C1.0	87.48 %	0.73 %	48000	$504 \times \Delta + 1$
BDT5-SVM-linear-C1.0	88.52 %	0.71 %	96000	$505 \times \Delta + 1$
BDT6-SVM-linear-C1.0	89.48 %	0.72 %	192000	$506 \times \Delta + 1$
BDT7-SVM-linear-C1.0	90.21 %	0.79 %	384000	$507 \times \Delta + 1$
BDT8-SVM-linear-C1.0	90.80 %	0.89 %	768000	$508 \times \Delta + 1$
BDT9-SVM-linear-C1.0	91.01 %	0.98 %	1536000	$509 \times \Delta + 1$
BDT10-SVM-linear-C1.0	91.13 %	0.96 %	3072000	$510 \times \Delta + 1$
BDT3-SVM-linear-C10.1	82.18 %	1.04 %	24000	$503 \times \Delta + 1$
BDT4-SVM-linear-C10.1	87.57 %	0.74 %	48000	$504 \times \Delta + 1$
BDT5-SVM-linear-C10.1	88.61 %	0.75 %	96000	$505 \times \Delta + 1$
BDT6-SVM-linear-C10.1	89.54 %	0.81 %	192000	$506 \times \Delta + 1$
BDT7-SVM-linear-C10.1	90.05 %	1.02 %	384000	$507 \times \Delta + 1$
BDT8-SVM-linear-C10.1	90.20 %	1.30 %	768000	$508 \times \Delta + 1$
BDT9-SVM-linear-C10.1	90.64 %	1.26 %	1536000	$509 \times \Delta + 1$
BDT10-SVM-linear-C10.1	91.02 %	1.03 %	3072000	$510 \times \Delta + 1$
BDT3-SVM-rbf-C0.1	76.80 %	0.48 %	487687288	$10237869 \times \Delta$
BDT4-SVM-rbf-C0.1	73.77 %	0.43 %	525479264	$5515615 \times \Delta$
BDT5-SVM-rbf-C0.1	70.57 %	0.42 %	566759360	$2974456 \times \Delta$
BDT6-SVM-rbf-C0.1	67.17 %	0.45 %	608135680	$1595807 \times \Delta$
BDT7-SVM-rbf-C0.1	63.10 %	0.45 %	655092992	$859518 \times \Delta$
BDT8-SVM-rbf-C0.1	59.41 %	0.48 %	715906560	$469659 \times \Delta$
BDT9-SVM-rbf-C0.1	53.54 %	0.41 %	783648256	$257055 \times \Delta$
BDT10-SVM-rbf-C0.1	45.37 %	0.31 %	856008704	$140401 \times \Delta$
BDT3-SVM-rbf-C1.0	85.19 %	0.43 %	332189304	$6973549 \times \Delta$
BDT4-SVM-rbf-C1.0	84.56 %	0.42 %	351073264	$3684990 \times \Delta$
BDT5-SVM-rbf-C1.0	83.78 %	0.41 %	373637792	$1960921 \times \Delta$
BDT6-SVM-rbf-C1.0	82.54 %	0.38 %	398559808	$1045862 \times \Delta$
BDT7-SVM-rbf-C1.0	81.11 %	0.37 %	430313728	$564598 \times \Delta$

Algorithm	TPR	FPR	MF (bytes)	ET(No. Operations)
BDT8-SVM-rbf-C1.0	79.28 %	0.37 %	470342912	$308564 \times \Delta$
BDT9-SVM-rbf-C1.0	77.17 %	0.37 %	518839808	$170195 \times \Delta$
BDT10-SVM-rbf-C1.0	74.63 %	0.38 %	581962752	$95456 \times \Delta$
BDT3-SVM-rbf-C10.1	90.57 %	0.33 %	241281680	$5065154 \times \Delta$
BDT4-SVM-rbf-C10.1	90.25 %	0.35 %	251770096	$2642670 \times \Delta$
BDT5-SVM-rbf-C10.1	90.08 %	0.36 %	262402848	$1377141 \times \Delta$
BDT6-SVM-rbf-C10.1	89.77 %	0.37 %	274623296	$720642 \times \Delta$
BDT7-SVM-rbf-C10.1	89.54 %	0.38 %	294830336	$386838 \times \Delta$
BDT8-SVM-rbf-C10.1	89.25 %	0.37 %	324082432	$212614 \times \Delta$
BDT9-SVM-rbf-C10.1	88.88 %	0.37 %	364881408	$119695 \times \Delta$
BDT10-SVM-rbf-C10.1	88.43 %	0.36 %	421846016	$69196 \times \Delta$
BDT3-CAS	59.94 %	0.109 %	480000	$10003 \times \Delta + 20$
BDT4-CAS	70.07 %	0.261 %	960000	$10004 \times \Delta + 20$
BDT4-CAS-3000	76.46 %	0.508 %	960000	$10004 \times \Delta + 20$
BDT4-CAS-200F-200IT	83.10 %	0.590 %	2688000	$28004 \times \Delta + 20$
CAS-BDT1-200F-200IT	84.02 %	0.869 %	3066000	$8008 \times \Delta + 16$

Table 4: Performance indicators obtained in practical experiments for all studied algorithms

[1] Dragos Gavrilut, Razvan Benchea, and Cristina Vatamanu. Optimized zero false positives perceptron training for malware detection. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012, Timisoara, Romania, September 26-29, 2012*, pages 247–253, 2012. doi: 10.1109/SYNASC.2012.34. URL https://doi.org/10.1109/SYNASC.2012.34.

[2] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 386–407, 1958.

[3] Stanley Ng Kwang Loong and Santosh K. Mishra. De novo SVM classification of precursor microRNAs from genomic pseudo hairpins using global and intrinsic folding measures. *Bioinformatics/computer Applications in The Biosciences*, 23:1321–1330, 2007. doi: 10.1093/bioinformatics/btm026.

[4] http://scikit-learn.org/stable/modules/svm.html#.