DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA E MATEMATICA APPLICATA

Master's degree in Artificial Intelligence and Intelligent Robotics

Report Artificial Intelligence for Cybersecurity

Implementation and validation of malware detection system

Lecturer:

Greco Antonio

Students:

Mariniello Alessandro	0622702106
Pepe Paolo	0622702005
Rabasca Dario	0622702003
Vicidomini Luigi	0622701949

Academic year 2023/2024

Contents

1 Problem analysis

The problem under examination involves the development and evaluation of the robustness of a malware detection system using machine learning and deep learning methodologies. For the training phase, as required, the training set and validation set from dataset 1 (with samples taken from the SoReL-20M dataset) were used, while the test set derived from the same dataset was used for the testing phase. To verify the models' generalization capabilities, their performance was also measured on the "VirusShare-Dataset". Additionally, an analysis of the robustness of the best models identified in the previous phase was performed, selecting one based on machine learning and one on deep learning. Subsequently, a discussion was conducted on possible defense methods that could improve the network's performance in case of an attack.

2 Selection of models

In the following chapter, the machine learning and deep learning models used in the experiments conducted within this project are detailed.

2.1 Choice of machine learning models

For training machine learning models based on EMBER features extracted from various files, several models were analyzed, each trained using a RandomGridSearch (a variant of GridSearch). The models considered are:

- Random Forest;
- AdaBoost;
- LightGBM;
- XGBoost;
- Ensemble of the aforementioned models.

For each model category, a grid of potential parameters was selected to ensure a trade-off between performance and training time. The parameters chosen for each model will be detailed in the chapter dedicated to the analysis of the machine learning models.

2.2 Choice of deep learning models

For the malware detection component using deep learning networks, three architectures examined during the course were considered: MalConv2, MalConvGCG, and AvastConv. Various configurations of the aforementioned models were tested by varying the following parameters:

- **Batch size:** for each network, training was conducted with batch sizes of 32, 64 and 128;
- **Hyper-parameters:** the networks' hyperparameters were adjusted to analyze their impact during the training phase.

Further details will be provided in the chapter dedicated to the training results.

3 Description of datasets

As mentioned in the introductory chapter, the datasets used in this study are as follows:

- **SoReL-20M:** utilized in the training, validation, and testing phases of the model;
- **VirusShareDataset:**employed to evaluate the generalization capabilities of the developed models.

In the following paragraphs, a detailed description of each dataset will be provided, including the main characteristics and data composition.

3.1 SoReL-20M

The SoReL-20M (Sophos/ReversingLabs-20 Million) dataset is a large-scale dataset comprising nearly 20 million files with pre-extracted features and metadata. High-quality labels are derived from multiple sources, providing detailed information on the detections made by various vendors for the malware samples at the time of collection. This dataset offers a solid foundation for training and evaluating malware detection models due to its breadth and the richness of the information contained. In the specific context of this study, however, the dataset was reduced to the following subdivisions:

- **Training set:** consisting of 9,196 samples, equally distributed between malware and benign files, with 4,598 samples of each type;
- **Validation set:** comprising 2,626 samples, also equally divided between malware and benign files, with 1,313 samples for each group;
- **Test set:** consisting of 1,312 samples, maintaining the same equal distribution between malware and benign files, with 656 samples for each category.

3.2 VirusShare Dataset

The VirusShare Dataset is a vast collection of malware samples used for cybersecurity research and the development of malware detection models. With millions of samples from various sources, it represents a fundamental resource for evaluating and improving detection model effectiveness. In this project, 2,000 samples were used, equally distributed between malware and benign files. The VirusShare Dataset was used to test the generalization capabilities of the developed malware detection models. Thanks to its wide variety of samples, this dataset serves as an ideal proving ground for assessing the robustness and reliability of the models in real and unpredictable scenarios.

4 Model training and validation

4.1 Machine learning model

4.1.1 Preliminary notes

Before analyzing the trained machine learning models, it is important to discuss the Ember features used for training. The process began with a simple training of the Random Forest and AdaBoost models using all available features. Analyzing the results on the test set, it was observed that the performance was perfect. Consequently, an analysis was conducted to identify the discriminating features among those available, and it was noted that one feature in particular (***coff-machine***) allowed the model to perfectly distinguish a benign file from malware. Analyzing the values of this feature, it was found that all malware samples lacked this information (it had the value UNKNOWN), while benign files had this feature. Subsequently, further training was conducted excluding the group to which this feature belonged. However, the same problem arose with another feature (***optional-subsystem***) that was not initially noticed because the training finished before it could be detected. Consequently, the group to which this feature belonged was also removed, and further training was conducted. The performance of the models trained this time was not perfect, but a feature (***coff-timestamp***) was found to have an importance level of 6%. It was decided to remove this feature as well, considering that the timestamp was irrelevant for malware detection. All subsequent trainings were conducted using a reduced version of the feature set, removing the previously discussed features from all training set samples. Therefore, the features used to train the various machine learning models were reduced from 2381 to 2360.

Trainings for the various models were conducted using a variant of Grid Search, namely Random Grid Search, which randomly selects a set of parameters from a grid of possible parameters. The parameters chosen for the various trainings were designed to ensure a reasonable training time and, at the same time, a model complexity that could guarantee good performance. The metric used by Grid Search to compare performance is accuracy. The number of folds was set to 5, an empirical K-fold value that allows for a proper training and validation split, enabling more accurate evaluations.

Additionally, despite a potential performance trade-off, it was decided not to use the validation set present in the SoReL-20M dataset, thus using only the training set for training the various models. This was done to allow for a comparative analysis between the machine learning and deep learning models obtained with approximately the same amount of data.

4.1.2 Parameters of Machine Learning Models

Below are listed, for each classifier, the respective parameters used during training[1]. The chosen parameters are based on training a portion of the SoReL-20M dataset. Therefore, it is presumed that the results on the SoReL-20M test set will be good for all trained

[1]Highlighted in bold are the parameters that provided the best performance following the different trainings

models, as the training set and test set have the same distribution.

4.1.3 Random Forest Classifier

For this model, the parameter grid used for the GridSearch is as follows:

- **Number of estimators**: **100**, 200, 300, 400, 500, 600, 700, 800, 900, 1000;
- **Max features**: **sqrt**;
- **Max depth**: **None**, 5, 8, 12, 16, 20;
- **Min sample split**: 2, 5, **10**;
- **Min samples leaf**: 1, **2**, 4.

4.1.4 LightGBM

For this model, the parameter grid used for the GridSearch is as follows:

- **Learning rate**: 0.01, **0.1**, 1;
- **Number of leaves**: 31, 127, **255**;
- **Reg alpha**: 0.1, **0.5**;
- **Min data in leaf**: 30, 50, **100**, 300, 400.

4.1.5 AdaBoost Classifier

For this model, the parameter grid used for the GridSearch is as follows:

- **Number of estimators**: 10, 50, 100, **500**;
- **Learning rate**: 0.001, 0.01, 0.1, **1**.

4.1.6 XGBoost

For this model, the parameter grid used for the GridSearch is as follows:

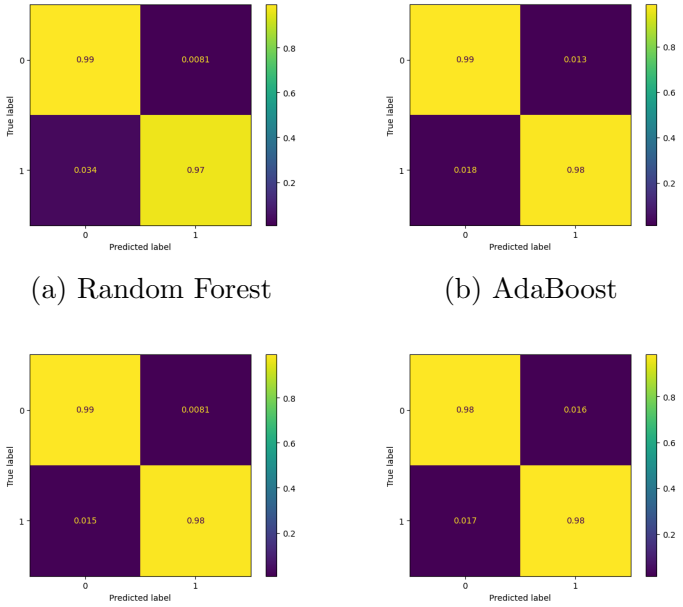
- **Learning rate**: 0.01, **0.1**, 1;
- **Number of estimators**: 50, **100**;
- **Max depth**: 10, 20, **30**.

4.1.7 Models performance on the Sorel-20M dataset

The performance on the SoReL-20M test set is as follows:

	Precision	Recall	F1-score	Accuracy score
Random Forest	0.98	0.98	0.98	0.98
AdaBoost	0.98	0.98	0.98	0.98
Light GBM	0.99	0.99	0.99	0.99
XGBoost	0.98	0.98	0.98	0.98

Table 1: Models metrics on SoReL-20M



(a) Random Forest (b) AdaBoost

(c) Light GBM (d) XGBoost

Figure 1: Models confusion matrices for SoReL-20M

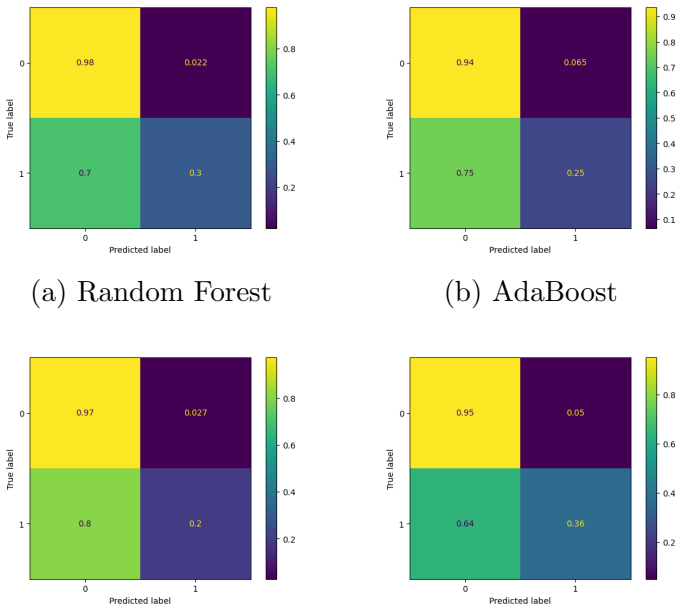
It is noted that all trained models have similar results among themselves on the SoReL-20M test set. This suggests that the dataset is well-balanced and contains representative information, allowing different models to learn effectively. Consequently, there are no significant differences in prediction capabilities among the models. Therefore, the selection of models for the Voting Classifier was based on performance on the second dataset (VirusShare).

4.1.8 Models performance on the VirusShare dataset

The performance on the VirusShareDataset test set is as follows:

	Precision	**Recall**	**F1-score**	**Accuracy score**
Random Forest	0.75	0.64	0.59	0.64
AdaBoost	0.67	0.59	0.54	0.59
Light GBM	0.71	0.58	0.51	0.58
XGBoost	0.74	0.66	0.62	0.66

Table 2: Models metrics on VirusShare



(a) Random Forest (b) AdaBoost

(c) Light GBM (d) XGBoost

Figure 2: Models confusion matrices for VirusShare

The XGBoost model appears to be the best, with an accuracy of 66% and a recall of 66%. This indicates that it can predict malware quite effectively compared to the other tested models. The Random Forest model also shows similar performance to XGBoost. However, Random Forest tends to classify more malicious files as benign, which is a problem because the main goal is to recognize as many malware as possible. Therefore, it is crucial to increase the classifier's sensitivity to improve the detection of malicious files.

4.1.9 Ensemble models

The ensemble models were chosen to combine the various previously trained models in order to improve performance, especially recall, thereby increasing the model's sensitivity. The models used for creating the ensembles include the Random Forest Classifier, Adaboost, and XGBoost, which achieved the best performance on the second test set, useful for understanding the models' generalization ability.

4.1.10 Parameters of the Ensemble models

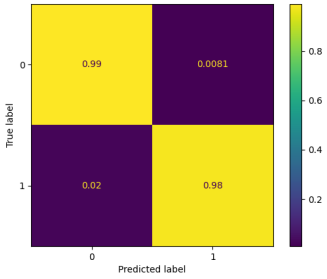
- **Random Forest Classifier:**
  - **Number of estimators:** 100
  - **Max features:** sqrt
  - **Max depth:** None
  - **Min sample split:** 10
  - **Min samples leaf:** 2
- **AdaBoost Classifier:**
  - **Number of estimators:** 500
  - **Learning rate:** 1.0
- **XGBoost:**
  - **Learning rate:** 0.1
  - **Number of estimators:** 100
  - **Max depth:** 30

4.1.11 Ensemble Models performance on the SoReL-20M dataset

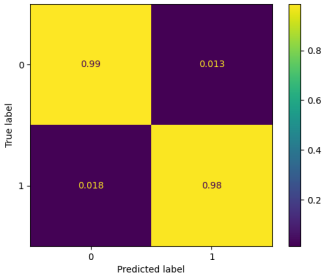
Testing on SoReL-20M Dataset:

	Voting	Precision	Recall	F1-score	Accuracy score
Random Forest, AdaBoost, XGBoost	Hard	0.98	0.99	0.99	0.99
Random Forest, AdaBoost, XGBoost	Soft	0.98	0.99	0.98	0.98
Random Forest, XGBoost	Hard	0.98	0.98	0.98	0.98
Random Forest, XGBoost	Soft	0.985	0.985	0.98	0.98

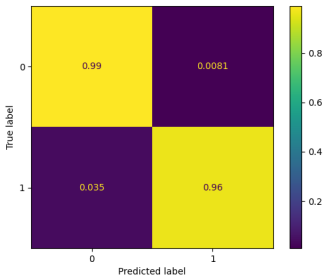
Table 3: Ensemble models metrics on SoReL-20M



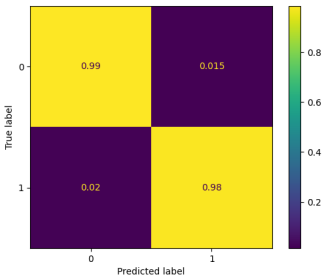
(a) Hard: Random Forest, AdaBoost e XGBoost



(b) Soft: Random Forest, AdaBoost e XGBoost



(c) Hard: Random Forest, XGBoost



(d) Soft: Random Forest, XGBoost

Figure 3: Performance of ensemble models on SoReL-20M

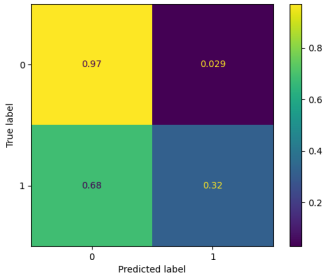
The models of the Voting Classifier exhibit similar performance to the individual models on the SoReL-20M dataset. However, a more interesting aspect is the comparison of results with the VirusShare dataset, which will be analyzed in the next paragraph, as it provides crucial insights into the models' generalization capability.

4.1.12 Ensemble models perfomance on VirusShare dataset

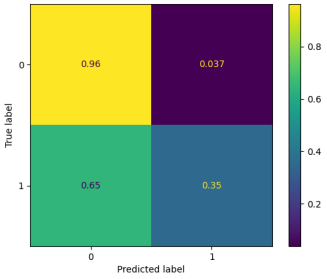
Testing on VirusShare Dataset:

	Voting	Precision	Recall	F1-score	Accuracy score
Random Forest, AdaBoost, XGBoost	Hard	0.75	0.645	0.60	0.64
Random Forest, AdaBoost, XGBoost	Soft	0.76	0.66	0.62	0.65
Random Forest, XGBoost	Hard	0.76	0.645	0.65	0.64
Random Forest, XGBoost	Soft	0.75	0.66	0.63	0.66

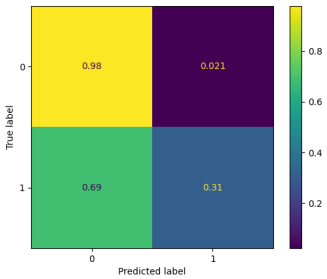
Table 4: Ensemble models metrics on VirusShare



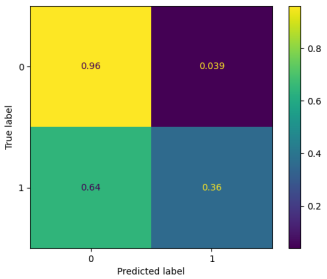
(a) Hard: Random Forest, AdaBoost e XGBoost



(b) Soft: Random Forest, AdaBoost e XGBoost



(c) Hard: Random Forest, XGBoost



(d) Soft: Random Forest, XGBoost

Figure 4: Performance of ensemble models on VirusShare

The best model is the Voting Classifier, which combines Random Forest and XGBoost using a soft voting scheme. Compared to the previous ensemble models, this model demonstrates higher accuracy and improved recall, resulting in a better ability to correctly detect malware, reducing the chances of misclassifying them as benign and,

consequently, mitigating the risk of infection. In the analysis, two Voting Classifiers were considered, excluding AdaBoost due to its inferior performance compared to Random Forest and XGBoost. This ensemble showed poorer performance compared to the aforementioned two models, confirming the superiority of the approach based on Random Forest and XGBoost. It was found that all Voting Classifier models with soft voting outperform their counterparts with hard voting. This improvement is attributed to the choice of soft voting, which takes into account the models' confidence levels in their predictions, thus leading to superior overall performance. Furthermore, all trained models excluding certain features show a decrease in performance on the SoReL-20M test set, but they improve their performance on the VirusShare dataset.

4.1.13 Model choice

All trained models ensure excellent performance on the SoReL-20M test set; however, their generalization capability is not entirely satisfactory. For future analyses, the chosen model is the Voting Classifier obtained by combining Random Forest and XGBoost with a soft voting scheme. This model was selected because it provided higher performance on the VirusShareDataset test set, with an accuracy and recall of 66%, despite its performance on the SoReL-20M test set, although high, not being the best among all trained models. Furthermore, from the analyses conducted previously, it is evident that Voting Classifier models are more effective in detecting malware in different contexts and scenarios not anticipated during the training phase, contributing to increased security and reliability in threat detection.

4.2 Deep learning model

As previously anticipated, several experiments were conducted to determine the optimal parameter setup to maximize model performance. In the following paragraphs, the results of these experiments will be presented, organized by network type and parameter type, accompanied by relevant considerations.

4.2.1 MalConv2

The conclusions drawn from the experiment conducted on MalConv2 demonstrate the critical importance of hyperparameters in determining the model's performance. Initially, two different configurations were explored while keeping the batch size constant at 32.

- The first training was launched with:
  - **channels** = 128;
  - **window size** = 500;
  - **stride** = 500;
  - **embedding size** = 8;
- The second training was launched with:
  - **channels** = 256;
  - **window size** = 256;
  - **stride** = 64;
  - **embedding size** = 8;

The results on the test sets are as follows:

	Training 1	**Training 2**
Accuracy SoRel-20M	92.98%	97.33%
Accuracy VirusShareDataset	68.55%	57.35%

Table 5: MalConv2 results

An interesting trend is noticeable in the results, where Training 2 shows better performance on Test Set 1 (SoReL-20M), but experiences a greater degradation of performance during the generalization phase compared to Training 1. This suggests that while it's possible to optimize models for specific performance on training data, it's essential to balance such configurations to ensure good generalization ability across different datasets. Having identified the best hyperparameters (those from the first training), we proceeded to evaluate batch size. The results are as follows (note that for batch size 32, the results obtained from the previous experiment were used):

From the data emerged, it can be concluded that training with a batch size of 32 demonstrates the best generalization capability compared to the other configurations

	Batch size 32	**Batch size 64**	**Batch size 128**
Accuracy SoReL-20M	92.98%	94.05%	95.5%
Accuracy VirusShareDataset	68.55%	64.2%	56.7%

Table 6: MalConv2 results with different batch size

tested, despite being less performant on the test set of the SoReL-20M dataset. This phenomenon suggests that the excellent results obtained by the other trainings on dataset 1 could be attributed to a case of overfitting, where the models are overly adapted to the specific characteristics of the samples present in that specific dataset.

4.2.2 MalConv GCG

Similarly to what was done for MalConv2 [4], different configurations were also explored for MalConvGCG[4], varying both the hyperparameters and the batch size. In particular, with the batch size fixed at 32, two separate trainings were conducted.

- The first training was launched with:
  - **channels** = 128;
  - **window size** = 512;
  - **stride** = 512;
  - **embedding size** = 8;
- The second training was launched with:
  - **channels** = 256;
  - **window size** = 256;
  - **stride** = 64;
  - **embedding size** = 8;

The results on the test sets are as follows:

	Training 1	**Training 2**
Accuracy SoRel-20M	93.29%	95.73%
Accuracy VirusShareDataset	61.3%	55.35%

Table 7: Risultati MalConvGCG

In this case as well, therefore, the first training yielded worse results on the SoReL-20M dataset but showed better generalization capability compared to the second training. This is why the decision was made to use the hyperparameters from the first training for the subsequent analysis. The subsequent analysis focused on the batch size to be used during training. The results are as follows (note that for batch size 32, the results obtained from the previous experiment were used):

	Batch size 32	**Batch size 64**	**Batch size 128**
Accuracy SoReL-20M	93.29%	91.92%	93.75%
Accuracy VirusShareDataset	61.3%	66.55%	56.8%

Table 8: MalConvGCG results with different batch size

Unlike what was observed for MalConv2, in this case, the best batch size among those examined is 64. It is noteworthy that, even in this situation, the model showing lower performance on the first dataset (SoReL-20M) turns out to be the best on the second dataset (VirusShareDataset). This phenomenon further emphasizes the importance of model generalization capability, suggesting that high accuracy on a specific training dataset does not always guarantee optimal performance on previously unseen data.

4.2.3 AvastConv

Finally, regarding AvastConv [1], the results of the initial trainings were not satisfactory, similar to what was observed for MalConv2 and MalConvGCG. Therefore, it was decided not to invest further resources in exploring different hyperparameter and batch size configurations. Below are the performances obtained using a batch size of 32 and the following hyperparameters:

- **channels** = 48;
- **window size** = 32;
- **stride** = 4;
- **embedding size** = 8;

	Batch size 32
Accuracy SoReL-20M	94.51%
Accuracy VirusShareDataset	58.9%

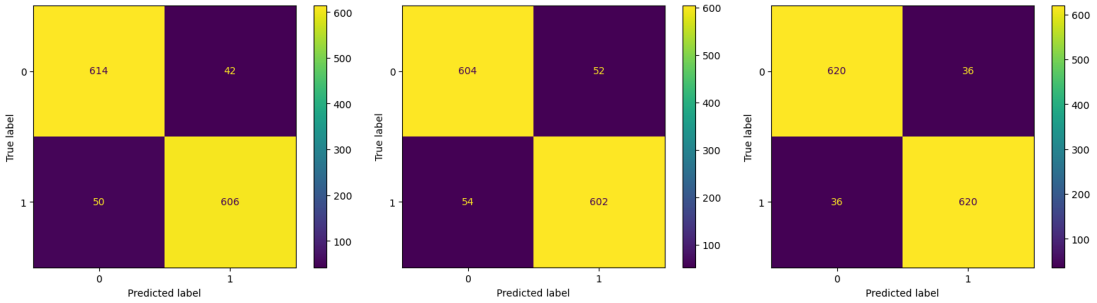
Table 9: AvastConv results

4.2.4 Models performance on SoReL-20M

It is evident that among the three models, AvastConv demonstrates the best overall performance. However, the previous experiments indicate that these high performances might suggest a risk of overfitting the model to the training data. Below are the confusion matrices of the models:

	MalConv2	**MalConvGCG**	**AvastConv**
Precision	0.93	0.92	0.94
Recall	0.92	0.91	0.94
F-Score	0.92	0.91	0.94

Table 10: Models metrics on SoReL-20M



(a) MalConv2 (b) MalConvGCG (c) AvastConv

Figure 5: Model's confusion matrices for SoReL-20M

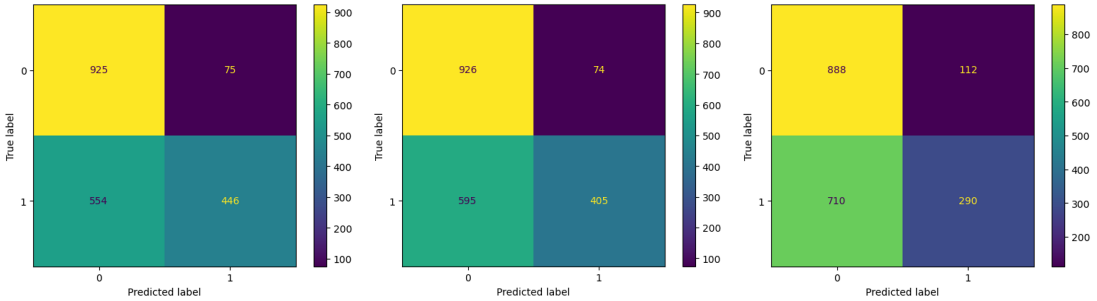
4.2.5 Models performance on VirusShare Dataset

Here are the values of the Precision, Recall, and F-Score metrics related to the SoReL-20M dataset for the best models identified earlier for each network type:

	MalConv2	**MalConvGCG**	**AvastConv**
Precision	0.85	0.84	0.72
Recall	0.44	0.40	0.29
F-Score	0.58	0.54	0.41

Table 11: Models metrics on VirusShare Dataset

It is immediately noticeable that compared to the previous metrics, the current values are significantly reduced. This result is reasonable considering that the data in this test set have a different format than those used for training the model. Below are the confusion matrices of the three models:



(a) MalConv2 (b) MalConvGCG (c) AvastConv

Figure 6: Model's confusion matrices for VirusShareDataset

5 Robustness evaluation of trained models

To evaluate the robustness of the trained models against potential attacks, GAMMA attacks were conducted, varying different parameters. Specifically:

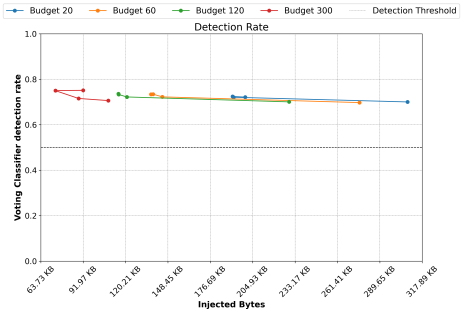
- $\boldsymbol{\lambda} = 10^{-3}, 10^{-5}, 10^{-7}, 10^{-9}$;
- ***how_many*** = 25, 50;
- ***query_budget*** = 20, 60, 120, 300.

This methodology allowed for a detailed examination of the models' resilience under varying attack conditions, ensuring a thorough evaluation of their performance under different stress configurations. In executing the GAMMA attacks, the model, both machine learning and deep learning, that demonstrated the best performance on the VirusShare-Dataset was selected. This approach ensured that the robustness analysis was conducted on the most effective models available, thereby providing a rigorous and accurate assessment of their ability to withstand compromise attempts.

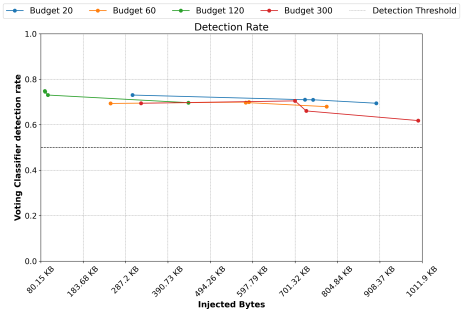
5.1 Machine learning model evaluation

Below are the results obtained for each attack configuration conducted on the best machine learning model, divided according to the *"how_many"* values used. Subsequently, the results obtained with *"how_many"* equal to 25 will be compared with those obtained with *"how_many"* equal to 50, in order to highlight the differences in performance and robustness of the models under examination. The model selected for the attacks is a voting classifier obtained by combining Random Forest and XGBoost models with a soft voting type.

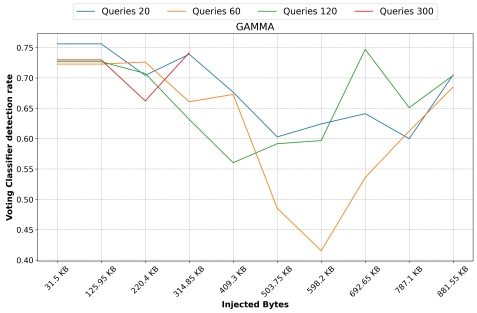
5.1.1 GAMMA attack evaluations



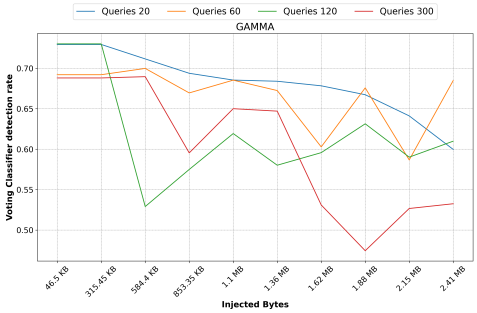
(a) How many 25 - Injected Bytes vs Detection Rate varying Query Budget and Lambda



(b) How many 50 - Injected Bytes vs Detection Rate varying Query Budget and Lambda



(c) How many 25 - Injected Bytes vs Detection Rate varying Query Budget

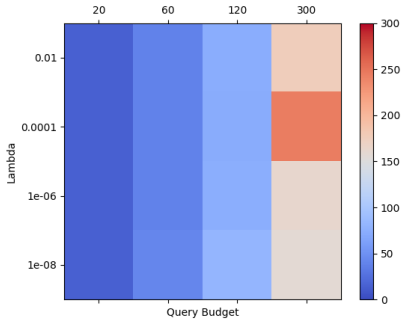


(d) How many 50 - Injected Bytes vs Detection Rate varying Query Budget

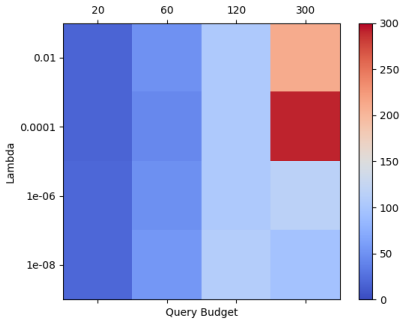
Figure 7: Graphs for machine learning models

For evaluating the robustness of the models, we chose to attack with two different values of *how_many* to understand the trend as this parameter varies. The machine learning models demonstrate robustness, achieving a detection rate of 0.7 for *how_many* 25, while it drops to approximately 0.6 for *how_many* 50. Regarding the injected bytes, there is a difference between the two values of *how_many*. Specifically, with *how_many*=25, we achieve practically the same decrease in the detection rate while injecting fewer bytes into the malware. Observing the second set of graphs, we note that with lower values of query budget, *how_many*, and injected bytes on average, we manage to bring the detection rate below 40%. As final observations, it can be stated that as the value of *how_many* increases, the detection rate decreases, but it injects many more bytes into the malware. Furthermore, it requires a much higher query budget compared to the first *how_many* to achieve those performance levels.

The 8a and 8b graphs illustrate the average complexity level of the solution found as the query budget and lambda values vary. Here, complexity refers to the average time used to create the corrupted sample in each of the possible configurations. Specifically, in the 8a graph, corresponding to a *how_many* value of 25, the complexity of the solutions obtained increases with the query budget. An interesting observation is that the combi-



(a) How many 25 - Injection complexity varying Query Budget and Lambda



(b) How many 50 - Injection complexity varying Query Budget and Lambda

Figure 8: Heatmaps for machine learning models

nation of a query budget of 300 and lambda 10^{-5} exhibits higher complexity compared to the query budget 300 column, as evident in the first graph. A similar pattern can be observed for the second heatmap, which also shows increasing complexity as the query budget value rises. However, the solutions found in the last column (query budget 300) with lambda values of 10^{-7} and 10^{-9} are simpler compared to the first heatmap. This is likely due to the attack having a wider range of benign sections to add and a smaller regularization term, resulting in finding a good configuration of sections to inject in a shorter time.

Results *how_many* = 25:

	$\lambda = 10^{-3}$	$\lambda = 10^{-5}$	$\lambda = 10^{-7}$	$\lambda = 10^{-9}$
Mean Elapsed time (s)	176.64, std: 120.06	244.60, std. 186.55	162.32, std: 116.08	157.49, std: 125.76
Mean Queries	296.0, std: 22.80	299.0, std: 8.18	262.0, std: 53.17	185.0, std: 60.95
Sum stagnated files	4	4	48	93
Mean of final classification	0.75, std: 0.08	0.74, std: 0.09	0.64, std: 0.16	0.67, std: 0.12
File that evade classification	3/100	4/100	22/100	10/100
Mean of injected bytes (KB)	91.96, std: 35.08	55.02, std: 24.78	119.83, std: 41.58	168.12, std: 32.27

Table 12: How many 25 - summary of attacks

Results *how_many* = 50:

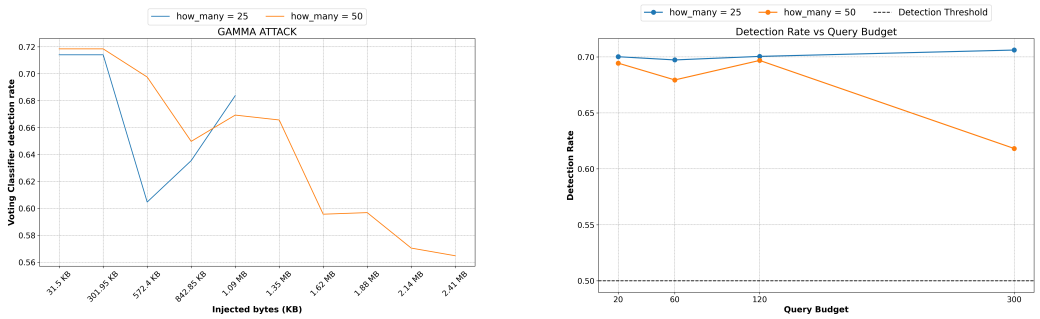
	$\lambda = 10^{-3}$	$\lambda = 10^{-5}$	$\lambda = 10^{-7}$	$\lambda = 10^{-9}$
Mean Elapsed time (s)	210.1, std. 124	291.5, std. 124.1	115.1, std. 58.42	95.71, std. 57
Mean Queries	299.2, std: 7.95	297.8, std: 11.27	292.4, std: 20.93	206, std: 63.15
Sum stagnated files	2	5	19	86
Mean of final classification	0.69, std: 0.13	0.71, std: 0.1	0.57, std: 0.15	0.49, std: 0.15
File that evade classification	11/100	6/100	35/100	54/100
Mean of injected bytes (KB)	324.95, std: 93.98	1050, std: 201	783.46, std: 133.63	1780, std: 373.02

Table 13: How many 50 - summary of attacks

In the tables, it can be seen that as the number of bytes injected increases, the number of samples evading classification also increases. The same behavior is present as the lambda value decreases, since the cost term is weighted less and consequently more complex solutions can be used.

5.1.2 Comparison of the two attacks

The 9a graph illustrates how the detection rate varies with the number of bytes injected for attacks conducted using *how_many* equal to 25 and 50. In the *how_many* 25 configuration, a U-shape pattern is observed, indicating that the attack improves (detection rate decreases) up to 572.4 KB, then deteriorates as the number of bytes injected further



(a) Detection rate vs injected byte

(b) Detection rate vs query budget

Figure 9: Comparison graphs for machine learning attacks

increases. This suggests that the model can distinguish a corrupted sample if the number of injected bytes increases excessively, likely due to the solution being found with a relatively small number of possible combinations of injectable sections. In contrast, for the *how_many* 50 configuration, the detection rate decreases almost linearly as the number of bytes injected increases. This behavior is attributed to the larger number of usable sections, resulting in a wider range of possible combinations for creating corrupted samples.

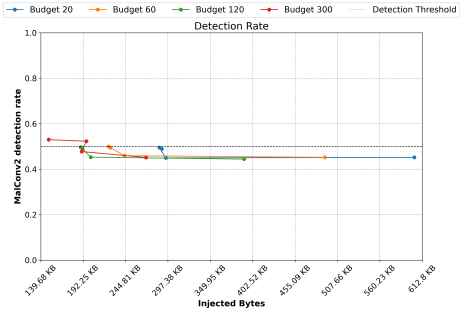
The 9b graph displays the detection rate as the query budget values vary for both *how_many* 25 and 50 configurations. In the *how_many* 25 configuration, the average detection rate for all query budgets settles close to 70 percent. However, there is an unusual behavior observed for query budgets of 300, where the resulting detection rate is higher than the previous ones. This behavior can be attributed to the size of the injectable sections. Regarding the *how_many* 50 configuration, as the query budget values increase, the detection rate decreases until it reaches a value close to 62%. This is also evident in the number of corrupted samples that fooled the model, with 54 out of 100 being successful.

5.2 Deep learning model evaluation

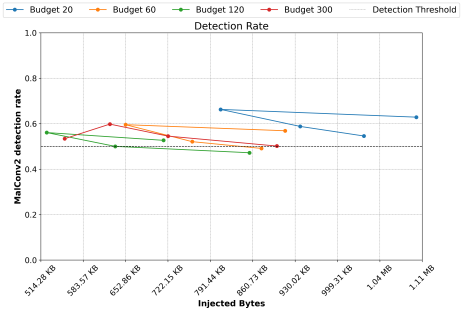
Following the same approach taken for the evaluation of GAMMA attacks on machine learning models, this section will present a detailed analysis of attacks conducted with different values of the *"how_many"* parameter. Specifically, the attacks will be examined separately for *"how_many"* values of 25 and 50. Then, the results obtained with these two configurations will be compared. The model selected for the attacks is MalConv2 with 32 of batch size and the standard parameters.

5.2.1 GAMMA attack evaluations

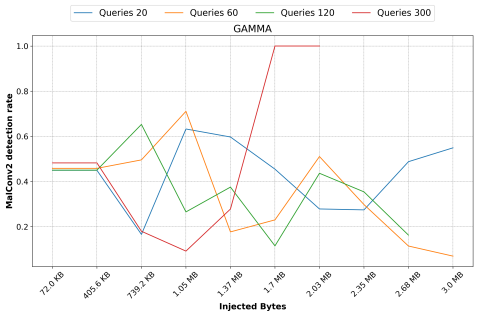
Regarding attacks on deep models, they prove to be much more effective compared to those conducted on handcrafted feature-based models, providing a higher number of corrupted samples that evade classification. For 10a, as the query budget value increases, the number of bytes injected into the samples decreases while the detection rate increases.



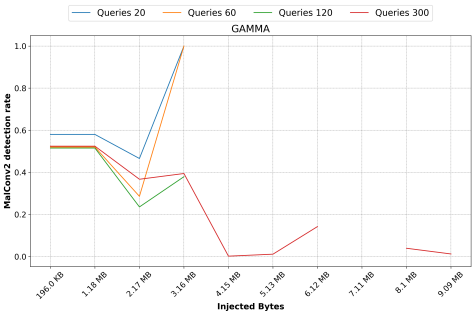
(a) How many 25 - Injected Bytes vs Detection Rate varying Query Budget and Lambda



(b) How many 50 - Injected Bytes vs Detection Rate varying Query Budget and Lambda



(c) How many 25 - Injected Bytes vs Detection Rate varying Query Budget

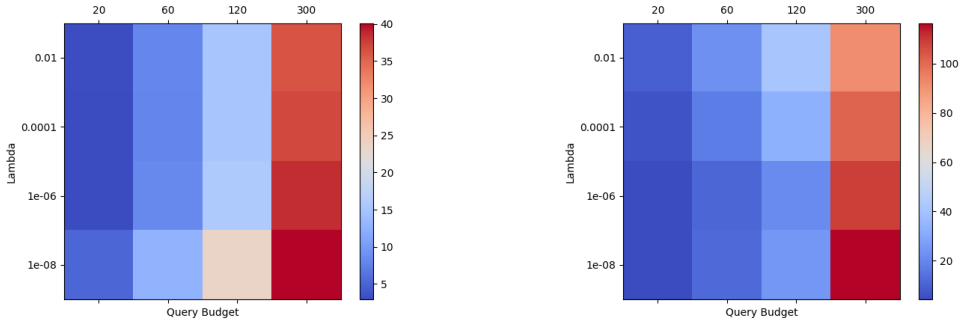


(d) How many 50 - Injected Bytes vs Detection Rate varying Query Budget

Figure 10: Graphs for deep learning models

This could be attributed to the deep model being deceived as more bytes are inserted into the sample. In this scenario, the model is better able to distinguish modified samples using solutions that are complex yet lightweight in terms of injected bytes, rather than those obtained using simpler yet heavier solutions. As for 10b, the results obtained are different from the expected ones. Indeed, for all query budget and lambda values, there is a detection rate that is almost always above the threshold value (set at 0.5). In this case, the average number of bytes injected is significantly higher than that in 10a, leading to worse attack performance. 10c demonstrates that samples generated with configurations having query budgets of 60 and 120, as the number of injected bytes increases, manage to deceive the model better, resulting in a detection rate below 20%. A similar behavior is observed for the configuration with a query budget of 20, with detection rate values fluctuating between 0.4 and 0.6 as the injected bytes vary. With a query budget of 300, the detection rate decreases until reaching 1.05 MB before rising to 1. 10 illustrates that samples generated with a query budget of 300 are able to lower the detection rate as the number of bytes used increases, while for higher query budgets, the network is able to recognize the presence of malware.

The two heatmaps illustrate the complexities of the solutions found in the two configurations as the query budget and lambda value vary. Here, complexity refers to the average time taken to find the solution in the various configurations. Regarding 11a,



(a) How many 25 - Injection complexity varying Query Budget and Lambda

(b) How many 50 - Injection complexity varying Query Budget and Lambda

Figure 11: Heatmaps for deep learning models

the complexity of the solutions increases as the query budget increases, with the highest complexity observed in the last column (query budget 300). The maximum value appears in the configuration with a query budget of 300 and lambda equal to 10^{-9}. This can be explained by the fact that, with this regularization term, the solutions were not constrained, allowing for more complex solutions to be found. A similar pattern is observed in 11b, where the complexity of solutions also increases with the query budget. However, in this case, the maximum value achievable is around 100, compared to 40 in 11a. This indicates that the solutions found with this configuration are much more complex. Despite the higher complexity of the solutions, the performance of the attack with this configuration was worse than the previous one.

Results *how_many* = 25:

	$\lambda = 10^{-3}$	$\lambda = 10^{-5}$	$\lambda = 10^{-7}$	$\lambda = 10^{-9}$
Mean Elapsed time (s)	36.2, std. 29.67	36.96, std. 28.43	38.6, std. 31.54	40.12, std. 30.15
Mean Queries	291, std: 28.9	281, std: 47	281.4, std: 43.1	255, std: 63.2
Sum stagnated files	15	18	21	49
Mean of final classification	0.52, std: 0.43	0.51, std: 0.43	0.38, std: 0.45	0.37, std: 0.45
File that evade classification	46/100	47/100	61/100	62/100
Mean of injected bytes (KB)	149.44, std: 160.52	243.01, std: 159.23	178.88, std: 166.4	509.35, std: 299.73

Table 14: How many 25 - Summary of attacks

Results *how_many* = 50:

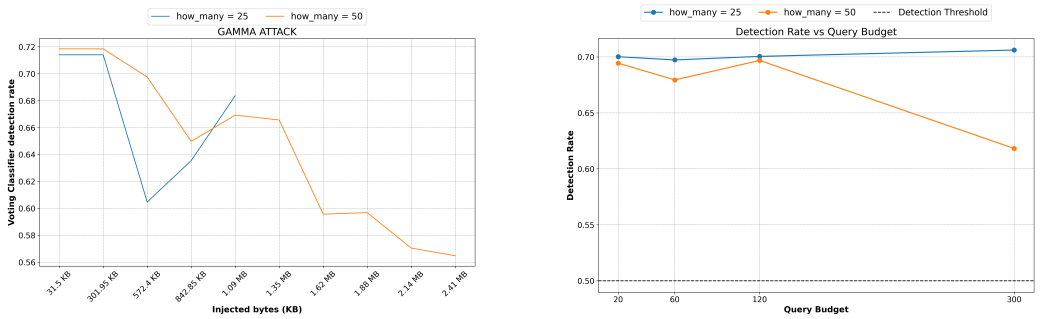
	$\lambda = 10^{-3}$	$\lambda = 10^{-5}$	$\lambda = 10^{-7}$	$\lambda = 10^{-9}$
Mean Elapsed time (s)	91.5, std. 54.1	101.32, std. 53.55	108.83, std. 57.81	116.46, std. 58.4
Mean Queries	291.8, std: 29.8	291.2, std: 29.57	282.4, std: 44.5	264.4, std: 58.3
Sum stagnated files	12	12	22	37
Mean of final classification	0.53, std: 0.42	0.66, std: 0.38	0.43, std: 0.43	0.37, std: 0.44
File that evade classification	46/100	29/100	55/100	60/100
Mean of injected bytes (KB)	553.34, std: 204.5	701, std: 236.98	912.9, std: 257.6	1400, std: 1650

Table 15: How many 50 - Summary of attacks

The results at varying lambdas show a reversal from the considerations made in the machine learning case. Specifically, the number of files evading classification is similar or slightly higher in the case with *how_many*=25, unlike the machine learning case where the performance at varying lambdas with *how_many*=25 was much lower than in the *how_many*=50 case. The advantage in this case of choosing *how_many*=25 is also that the average number of bytes injected is far less compared to *how_many*=50.

5.2.2 Comparison of the two attacks

The comparison between the two *how_many* values confirms the considerations made in the previous graphs. Specifically, in the first graph, we observe that on average the number of bytes injected is the same, but with *how_many*=25, we achieve a more linear behavior, whereas with *how_many*=50, the trend is less linear. The second graph further



(a) Detection rate vs injected byte

(b) Detection rate vs query budget

Figure 12: Comparison graphs for deep learning attacks

supports these observations by clearly showing that the detection rate for *how_many*=25 remains constant even as the *query_budget* varies, while for *how_many*=50, the detection rate is worse and less consistent as the *query_budget* varies.

5.3 Comparison of machine learning model and deep learning model

A fundamental observation regarding the robustness of the models is the greater effectiveness of machine learning models compared to deep learning models in the face of gamma attacks. As shown in 7a and 7b, the detection rate for machine learning models never falls below the threshold, whereas for deep learning models, the detection rate drops below 0.5. Considering that we performed the attacks with two different *how_many* values, some interesting insights can be drawn about their impact on the different models. Specifically, for the machine learning model, attacks are much more effective with *how_many*=50, as clearly shown in 9a. Conversely, with *how_many*=25, the model is less affected, as illustrated by the tables showing variations in lambdas. This is likely because with *how_many*=50, there are more byte combinations to inject, thus making the attack more successful. In contrast, deep learning models generally exhibit greater vulnerability to attacks, with detection rates falling below the 0.5 threshold in 10a and 10b. An interesting point is that these models are susceptible to attacks regardless of the *how_many* parameter value, as highlighted in the 14 and 15 tables. Another observation is that with *how_many*, the number of bytes injected decreases significantly.

Moreover, the results of the attacks confirm what had been seen in testing. The machine learning model has a recall that is about 20 percent higher than the deep model. This property made the machine learning model more robust than the deep learning model on the various attacks conducted. Thus, to fool the machine learning model there is a need to use a more complex combination to inject, this goes to explain that the performance of the machine learning model decreased as the value of *how_many* increased. The deep model, on the other hand has much lower recall; in fact, the various attacks conducted did not need very complex solutions to fool the model and, as a result, the parameter *how_many*=25 is also effective.

6 Analysis of possible defense methods

There are various strategies that can be adopted to improve the robustness of detection models, focusing on features that are less susceptible to manipulations or implementing advanced training and defense techniques. In the context of this paragraph, we will explore a variety of approaches aimed at increasing the resilience of malware detection systems against adversarial attacks, considering both specific features and machine learning techniques used to strengthen the models.

6.1 Features selection

To effectively address GAMMA attacks, it is crucial to consider features that are more resistant to such attacks. In particular, it is useful to focus on those features that do not undergo changes at the byte or file section level. However, to do this, it is essential to have a detailed understanding of the sections that are altered during the attack. For example, if it is known beforehand that a certain section or portion of a file remains unchanged, attention can be focused on these features for classification. This approach is preferable to using features that, although important, may have been compromised during the attack. Another example [3] involves using a graph-based representation, such as the Abstract Syntax Tree (AST). While this method incurs higher computational costs due to feature extraction, it offers greater resilience against manipulations introduced by GAMMA attacks.

6.2 Adversarial Training

Another potentially effective defense against attacks involves implementing Adversarial Training [3]. This approach entails training malware detection models using corrupted samples as well, allowing the network to learn which features are discriminative even in the presence of manipulated data. Additionally, to further enhance the model's robustness, it would be beneficial to train it using a larger and more diversified dataset containing examples of files with diverse structures. In our context, utilizing both the Sorel-20M and VirusShareDataset datasets could ensure superior generalization capabilities. This is because by adapting to files with different structures, the network would learn high-level representations that are difficult for attackers to manipulate. Another strategy could involve employing a combined model that leverages both features extracted from deep learning models and those obtained from Ember models. This approach would enable acquiring an even more detailed understanding of the sample, thereby improving the accuracy and robustness of the detection system.

6.3 Private handcrafted features

An additional approach to enhance the effectiveness of malware detection models involves the use of proprietary handcrafted features. This method entails the definition and creation, within the company, of unique and specific features that remain private. Handcrafted features, being specifically designed according to the needs and peculiarities

of the corporate environment, can provide greater precision in identifying malware. Implementing proprietary features allows the integration of knowledge and insights derived from experience and analysis of the organization's specific data into machine learning models. These features may include, for example, particular behavioral patterns, recurring anomalies, and other contextual information that is not immediately apparent in public datasets.

6.4 Feature squeezing

The concept of "feature squeezing" [2] represents an advanced defensive strategy against adversarial attacks, aimed at reducing the degrees of freedom that an attacker can exploit to create adversarial samples. This technique involves compressing or eliminating unnecessary features, thus narrowing the space in which attackers can operate. To determine whether a sample is malware, the model's predictions on the original sample and the sample obtained after the feature squeezing process are compared. If the distance between these predictions exceeds a predefined threshold, it can be concluded that the sample in question is likely malware. This is because a significant difference between the predictions indicates that the sample may contain manipulations typical of an attack. Adopting feature squeezing not only makes it more difficult for attackers to create effective adversarial samples but also increases the overall robustness of the malware detection model. By reducing the maneuvering space for manipulations, the model's ability to distinguish between legitimate and malicious data is improved, ensuring greater cybersecurity.

6.5 Dimensionality reduction

Dimensionality reduction is an effective defensive strategy in the context of malware detection and attacks. A commonly used method for such reduction is Principal Component Analysis (PCA). This approach allows an original dataset of size n to be transformed into a reduced set of size k (where k is much smaller than n) while retaining most of the relevant information. Instead of training a classifier on the entire original dataset, PCA is used to reduce the number of features. Next, the classifier is trained on the reduced input. This process not only simplifies the model but also limits the possibility of manipulation by attackers by focusing on the first k principal components. The assumption behind this defense is that adversary examples are based primarily on the principal components of the dataset. Therefore, any attack that aims to manipulate these principal components should result in a significant increase in the distortion required to produce adversarial examples. In other words, compressing the data into the first k components increases the difficulty for attackers to introduce imperceptible changes that can fool the model.

6.6 Query monitoring

In the context of black-box attacks, a malicious user can make repeated queries to the target model in a very short period [3]. This type of attack leverages continuous inter-

action with the model to gather information and generate effective adversarial samples. One possible defense strategy against such attacks is to analyze the number of queries from the same source. By carefully monitoring the frequency and volume of queries, it is possible to detect anomalous behaviors that suggest an attempted attack. For example, an unusually high number of queries in a short period could indicate an ongoing attack. Implementing a query monitoring system offers several advantages. Firstly, it allows for the identification and blocking of potential attacks before they can cause significant damage. This proactive approach enables timely intervention, limiting the attacker's chances of obtaining sufficient information to generate adversarial samples.

6.7 Continuous Training

Another strategy to enhance the security of malware detection systems involves using neural networks that support easy and continuous updating. This approach entails the ability to retrain models by adding new data containing examples of "new attacks" the network has not encountered before. This ensures the system's effectiveness in detecting emerging and evolving threats. Moreover, the extensive dataset used for training, incorporating both historical examples and new data, enables the model to develop a deeper understanding of malware features and behaviors. Such defense is a classic approach in cybersecurity contexts, where attackers constantly find new evasion methods, necessitating ongoing updates for defenders.

List of Figures

List of Tables

References

[1] Marek Krčál et al. “Deep convolutional malware classifiers can learn from raw executables and labels only”. In: (2018).

[2] Yonghong Huang et al. “Malware Evasion Attack and Defense”. In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 2019, pp. 34–38. DOI: 10.1109/DSN-W.2019.00014.

[3] Luca Demetrio et al. “Functionality-preserving black-box optimization of adversarial windows malware”. In: *IEEE Transactions on Information Forensics and Security* 16 (2021), pp. 3469–3478.

[4] Edward Raff et al. “Classifying sequences of extreme length with constant memory applied to malware detection”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 11. 2021, pp. 9386–9394.