DroidEvolver: Self-Evolving Android Malware Detection System - Summary - SidharthAnil

Presentation

The paper introduces a new Malware Detection system, DroidEvolver, to handle concept drift without the need for an explicit retraining stage or true labels. The system has a model pool consisting of 5 models and the premise is that the rate at which drift would affect each model would be different. Hence, when one model is observed to be failing in classifying a sample correctly due to concept drift, the results from the other models can be utilized to create a pseudo label with which the aging model would be updated. This would mean that the updation process takes place continuously throughout the functioning of the system without any human intervention and without the need for true labels. The models used in DroidEvolver's model pool are built using the following online learning algorithms:

1. Passive Aggressive - First Order learning that uses Hinge Loss and changing learning rate
2. Online Gradient Descent - First Order learning that uses an incremental approach with hinge loss but with fixed learning rate
3. Adaptive Regularization of Weight Vectors - Second-order learning, that updates the mean and covariance of Gaussian Distribution based on classification results of new samples.
4. Regularized Dual Averaging - Online learning with regularization that adjusts parameters for an optimization problem
5. Adaptive Forest Backward Splitting - Online learning with regularization that uses geometric information of the data distribution, unlike RDA.

Once the model pool is initialized, each time a new sample comes in a Juvenilization Indicator (JI) of that sample is calculated for each model. JI score is basically a similarity score of the new sample compared to a subset of samples the model has already seen. A buffer is kept for each model to store this subset of samples to be compared against.
If a sample is not drifting according to JI scored for any model, then a weighted voting is done across the models to finalize the classification. Moreover, the feature vector for this new sample could randomly replace a sample in the buffer.
If a sample is drifting for a set of models according to the JI score, a weighted voting is done across all the non-drifting models and the resulting classification is used as a pseudo label. The feature set of the aging models would be updated by including the features of the drifting sample, and the model structure would be updated using the drifting application and its pseudo label.
DroidEvolver's performance was compared against the then state-of-the-art malware classification system MAMADROID. It was found that DroidEvolver outperforms MAMDROID by a 15.8% F1 Score, 12.97% precision, and 17.57% recall on average. More importantly, it was observed that while the MAMDROID F-measure kept falling over a period of around 5 years, DroidEvolver's F-measure remained stable in comparison. It was also observed that the number of features used by DroidEvolver

kept on increasing over the years.
The presentation did point out that DroidEvolver is a static analysis system and hence cannot be compared to the power of a dynamic one, and moreover is susceptible to poisoning attack where the training dataset could be poisoned to affect the accuracy.

Discussion

- The first point of discussion was how DroidEvolver uses 5 models in its model pool and the practical consequences of such a strategy. Most devices or even the AV cloud server might not have the resources to accommodate 5 fully-fledged models and hence this points to a shortcoming of the paper in that it did not focus on such engineering aspects but remained for the most part academic and overly optimistic.
- Another aspect to be noted is that even though the dataset used by DroidEvolver for evaluation is fairly balanced, they have made no mention of the context of this model, the environment for which the model was designed, or the priorities it has. For this reason, it is hard to judge if the dataset used is representative.
- A question that arose was that if drift detection and subsequent retraining strategies are implemented at the cloud level, how does the cloud system identify drifts happening at the local client-side systems? It is normally detected using the data collected from the client side, including metadata as well as actual samples.
- A lot of malware detection systems use the classification by VirusTotal or such services for labeling the data. A question arose if this was a good practice, as it is essentially using the classifications a file acquires under the existing popular AntiVirus Engines and not necessarily the ultimately true label. This again is an engineering compromise made as it is not practical for a human malware analyst to go through every sample. Moreover, for the majority of the samples (excluding malware files using zero-day exploits) a labeling decision can be made using the results provided by services like VirusTotal.
- Moreover, the threshold for a file to be considered malicious based on the VirusTotal reports can be adjusted by the engineer depending on the requirements he has in mind. For example, this threshold would have a direct correlation to False Positive Rate. If only 1 out of 10 AV Engines need to identify a file as malicious, for it to be labeled malicious, it could lead to a higher FPR.
- Another point that was mentioned was that not all of the samples would be sent to the drift detection layer in order to utilize resources more efficiently. A much more practical strategy is for the samples that a first layer was not able to classify confidently to be sent to a second layer that involves drift detection.
- Regarding the increasing number of features utilized by DroidEvolver over the years, a question arose on shouldn't the redundant features be dropped as time progresses. The answer to this was that the situation is not always so straightforward as a feature that might be redundant at the moment, could very well be important in the future again as the malware trends or the underlying system architecture change.
- An important observation was made regarding the significance of temporal information. For example, a dataset might look fairly balanced on an overall scale, but when sorted and grouped

according to their timestamps, it might be totally imbalanced in specific time periods.

- There are different strategies to deal with concept drift and many drift detectors that can achieve this purpose. The best drift detector might be a point of debate or discussion and might depend entirely on the resources you have, and the type of data your model sees. However, it is fairly clear that a model that accommodates for concept drift and takes measures to rectify its consequences is better than a model that doesn't do this. This tallies with the observation of how MAMADROID's performance kept falling across the years, while DROIDEvolver's remained fairly stable.
- Building on the previous point, a point was raised that it is not fair to compare two such models as it would result in an Inappropriate Baseline pitfall. The student insisted that it would be a better comparison for DROIDEvolver to be pitted against a model, say, that retrains the model structure but not the feature space. As a response, it was noted that MAMADROID was a popular state-of-the-art model that was meant to persist in its performance across the year and hence the comparison is fairly justified.
- One aspect that the students focused on was the lack of human intervention in the whole process. Even though the benefits and convenience of such a model are apparent, the shortcomings of such a system were discussed. It was decided that a level of human intervention would be required to manually introduce specialized features that the automatic feature set updation could not handle. Moreover, the potential of poisoning the model becomes an even bigger threat with the lack of human intervention and it was agreed that even though it is not viable in an offline model it could be possible in an online one.
- This led to a discussion on backdoors that can be introduced in machine learning models. In computer systems, backdoors are secret routes that a malicious actor left behind so that he/she can gain access to the system easily in the future. In ML-based malware models, backdoors can be in the form of teaching the model to identify certain rare words (that would not be used in regular contexts) as benign. Later, the attacker could use these words in his malware files to evade the detector. It was also explicitly mentioned not to employ such techniques for the in-class competition/project as it would not be fair for the other teams.