Fast & Furious: Modelling Malware Detection as Evolving Data Streams - Summary - SidharthAnil

Paper

The paper mainly revolves around concept drift, its effects, and the ways to deal with this issue, particularly in the landscape of Android Malware. As opposed to the literature that was available at the time in this particular area, this paper introduces a novel approach of data stream machine learning-based malware model using a drift detector that retrains the static feature extractor along with the classifier in its pipeline.

Toward the end of creating a model, there are a couple of different choices to be made at various points in the machine learning workflow by the engineers. The type of drift detector to be used, the feature extraction strategy, etc are some examples of these decisions. Since each of these choices would have its own benefits and shortcomings, the authors have experimented and compared them using experimental data and have presented them systematically in the paper.

Firstly, in order to understand whether the issue of concept drift is related to the dataset in usage or if it's a general issue that should be combated, the authors used two different Android Malware datasets - DREBIN and a subset of AndroZoo (the AndroZoo dataset is more complicated and vast compared to DREBIN). Throughout the study, all experiments are done both on DREBIN and AndroZoo, and through this process, they drew the conclusion that the issue is persistent across datasets (even though in varying scales) and hence it should be considered irrespective of the dataset used in future research.

Since the core idea presented by the paper is an addition of retraining the feature extractor, a feature extractor that can be trained or derived from a dataset was necessary. For this purpose, the two candidates were Word2Vec and TF-IDF. After comparing the two extractors, using two different models (Random Forest and SGD), it was observed that TF-IDF performs better than Word2Vec. Hence, for the rest of the study, TF-IDF was utilized as the feature extractor.

Another key result presented in the paper is the performance of these models in the context of temporal classification. The dataset was split based on the timestamps into train and test sets, where the train set contains data from the past and the test set contains data from the future. The results of this experiment display a clear drop in the metrics and hence shows the need for a model that continuously learns, as opposed to a model that is trained and then left alone to detect malware forever into the future. Also, a notable observation is that across time, the goodware samples remain more or less similar, whereas the malware samples change drastically.

There are multiple ways in which a model can continuously learn. One way to achieve this is using an Incremental Windowed Classifier (IWC) where the model and the feature extractor are retrained once a fixed time interval has passed. Another strategy that could be employed is by using drift detectors. 4 different drift detectors were selected for this study, and they are:

1. Drift Detection Method (DDM)
2. Early Drift Detection Method (EDDM)
3. ADaptive WINdowing (ADWIN)
4. Kolmogorov-Smirnov WINdowing (KSWIN)

Within the strategy of using drift detectors, there are two possible design choices.

1. Update - Update the classifier alone with the new samples that were collected during the warning period of drift detectors.
2. Retrain - Both the classifier and the feature extractor are retrained from scratch using the data collected during the warning period of drift detectors.

So, in total 9 different models that the authors of the paper ran against the DREBIN and AndroZoo datasets for comparison. The results of this comparison indicated that retraining the model as well as the feature extractor using a drift detector (ADWIN or KSWIN) was found to be the most effective in terms of dealing with concept drift.

On analyzing concept drift itself, two main reasons were noted as the cause for it. One is the changing trend in malicious techniques, where the attacker constantly employs new strategies or focuses on new exploits depending on their effectiveness in the current scenario. Secondly, the android system, its underlying architecture, the features it presents to its users, etc keep changing, to which the attackers also adapt and evolve.

Discussion

- A key point of discussion was how the environment of Android Malware detection affects the strategies and defenses that could be employed. For example, since the Android endpoint devices are limited in resources, dynamic feature extraction or heavy models are not an option. This forces the researchers to come up with ways to engineer effective malware detection models that use only static features and could be run effectively within the resource limitation. At the same time, this encourages attackers to focus more on the Android Operating System because of its widespread usage as well as the limited power its AntiVirus systems would have.
- Building from the previous point, it could be seen how concept drift in the Android Malware landscape would be different when compared to other environments. Android is an operating system that has regular changes in its functionalities, the APIs exposed for the developers, and the features presented to the users. This would cause the whole data distribution of the malware files to change as attackers respond to this. For example, in the past SMS was a mechanism that was one of the major points of focus for attackers as it could be utilized to subscribe to services without the

user's knowledge or consent. But this changed when Android started restricting SMS usage especially for subscribing to services and hence causing a major concept drift. This type of concept drifts due to changes in the internal system can be observed in the vocabulary of the feature extractor, where words are dropped and new words are added correspondingly.

- Another major point of discussion was the way in which the features could be represented and how it would affect its representational power and consequently the model itself. This is where the concept of embedding (TF-IDF, Word2Vec, etc.) comes into the picture where semantic information between the words is also preserved which is valuable for detecting malware. Such types of semantic preserving embedding could be done along with simple encodings. For example, the attributes could be encoded using, say, one hot encoding and the result could then be embedded to contain the relations and patterns between the encoded values.
- Another form of representation that has even more power to store information is in the form of graphs, for example, control flow graphs or call graphs. Even though generally it would be hard to extract such graphs in a static manner, in the case of Android Apps this would be possible as it is built using languages like Java where such information is part of the language design. Another such graph that could be utilized is the Data Dependency Graph, which shows the dependent functions a particular function has.
- For an attacker, it is harder to evade a model that uses graphs as opposed to simpler or primitive embedding. Simply adding benign elements might not be enough as it need not cause enough change in the structure of the graph. Nevertheless, the attackers do manage to figure out evasions for graph embeddings too. For example, they might add dead code that will never be invoked during the run time. This would change the structure of the graph but at the same time wouldn't change the functionality of the file. Another example of a point of focus that the attackers exploit is exception handlers which would break the linearity of the control flow and cause the program flow to jump to another location. This could be used to add benign elements that will never be called upon during runtime, to fool the detector.
- The next topic of discussion was toward drift detectors and how often they should be called upon. For example, it is possible to build a model where the threshold for drift detection is low and hence would retrain very often. However, this would utilize much more resources when compared to a model that retrains itself less frequently. Again, this points towards the environment in which the model resides, the resource limitations of that environment, etc.
- Moreover, drift detection and retraining cannot happen synchronously. That is, it is not viable in most cases to stop the functioning of the entire model while the model is being retrained. For this reason, it could be designed to take place in an asynchronous manner in a different pipe behind the scenes, where a model is being retrained in a continuous manner. This strategy is done in a proactive manner where the possibility of a drift is predicted and retraining happens based on that as opposed to waiting for the drift to happen before initiating a retraining session.
- For such proactive drift detection you need ground-truth-like labels to compare your model's predictions against in order to predict the occurrence of drift. Since for the case of malware, this is hard to achieve immediately, one strategy employed is to run a different model in another pipe and use the labels predicted by this model as pseudo labels for comparison. This is not the case for active drift detection, as the dropping detection rate itself can be an indicator of the occurrence of

drift. As it can be obviously understood endpoint devices would not have the resources to employ such strategies and hence this is normally done in the cloud servers of the AntiVirus companies.